Spectral Coefficient Analysis on Geometerical Deformation using Laplace-Beltrami Operator

An Undergraduate Thesis
Submitted to the Department of Mechanical Engineering
The Ohio State University
In Partial Fulfillment of the Requirements for Graduation with Distinction in Mechanical Engineering

Skylar Sible

Defense Committe:
Dr. Shawn Midlam-Mohler, Advisor
Dr. Sheng Dong

April 16, 2020

Contents

List of Figures

Abstract

Automotive part design consists of crash simulation and design optimization done by finite-element computing software. In order to make these processes possible, the part geometry must be efficiently represented. Different parameters can be defined in an optimization in order to determine appropriate material models and part thickness, however there are still computational limitations on optimizing based on specific part deformation. This paper proposes a method that goes beyond tracking individual node displacements in order to use deformation as a parameter for optimized part design. Using a dimension reduction method, deformations of the part can be classified using a spectral descriptor corresponding to that deformation. This spectral descriptor is taken a step further and is used to efficiently filter a simulation bundle based on the defined desired geometric deformation. In addition, this spectral descriptor is used for part reconstruction with a higher visual accuracy compared to traditional reconstruction methods. Finally, this paper proposes application of this method into design optimization using a machine learning approach.

Keywords: design optimization, dimension reduction, part deformation, spectral descriptor

Acknowledgments

I would like to thank Shawn Midlam-Mohler for advising this research project. I would also like to thank Patricia Wollstadt and Nicola Aulig for their mentorship during my internship at HRI-EU, where this research began. I would like to thank Rodrigo Iza-Teran for allowing me to use the Dataviewer software at the onset of this research project. Lastly i would like to thank Emily Nutwell for allowing me to use part of her LS-Dyna hat section model for the simulations run in this thesis.

Chapter 1

Introduction

1.1 Background and motivation

Advancements of computational analysis have replaced physical crash testing. This has made a huge impact on the automotive industry, and will continue to improve the automotive design process, by reducing cost and time, and improving efficiency and safety. Finite element analysis is utilized in order to achieve all of these goals. When an engineer is designing a part, they must consider all the different solutions to accomplish their given goal. A central goal is achieving specific deformations under different crash scenarios, i.e. crashworthiness. This paper explores the need for engineers to design parts with very specific physical deformations in different crash scenarios.

In his book, Hesham Kamel Ibrahim defines crashworthiness as the process of improving the crash performance of a structure by sacrificing it under impact for the purpose of protecting occupants from injuries [3]. Crashworthiness can be improved by increasing the energy absorption and influencing the deformation mode of the part. This results in reducing the impact on the occupants and protects the other parts in the assembly.

Because the energy absorption is greatly effected by the deformation, influencing certain deformation modes can be a powerful design tool. The areas in the vehicle designed to absorb the energy, or load, of the impact can be defined as crumple zones. The crumple zones protect the crashworthiness of a vehicle by deforming in a way that optimizes the amount of energy absorbed and the severity of the damage to the rest of the vehicle and occupants. The two types of deformation focused on in this paper are axial deformation, also known as folding deformation, and bending deformation as shown in Figures 1.1 and 1.2.

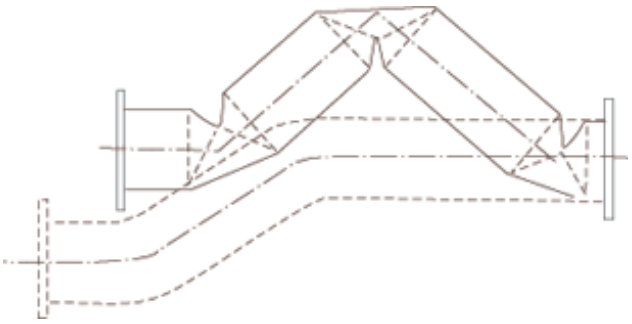

Figure 1.1: Bending deformation mode [1].

Figure 1.2: Axial or folding deformation mode [1].

When it comes to energy absorption in a crumple zone, a part undergoing an axial deformation is most effective. Due to instabilities it is difficult to achieve axial deformation, and often bending is the result of a failed axial deformation mode [1]. In other scenarios bending at a local hinge is desired to protect passengers, or other parts of the vehicle. Often this bending mode fails by a bending deformation occurring at an undesired local hinge. When structural parts are put under a compressive load, deformations can easily become irregular, or unstable, this causes a loss in energy absorption. A loss in energy absorption, may completely alter the crash behavior of the entire system. Examples of

these instabilities are shown in Figure 1.3, certain conditions must be met prior to a crash in order to achieve the desired deformation, slight differences in these conditions result in a high variance in crash results, in a complex system this will cause an undesired chain reaction effecting the entire model. Paul Du Bois stresses in his paper, 'Vehicle Crashworthiness and Occupant Protection', that the need to understand the triggers causing instabilities is critical in order to design countermeasures to prevent them [1].

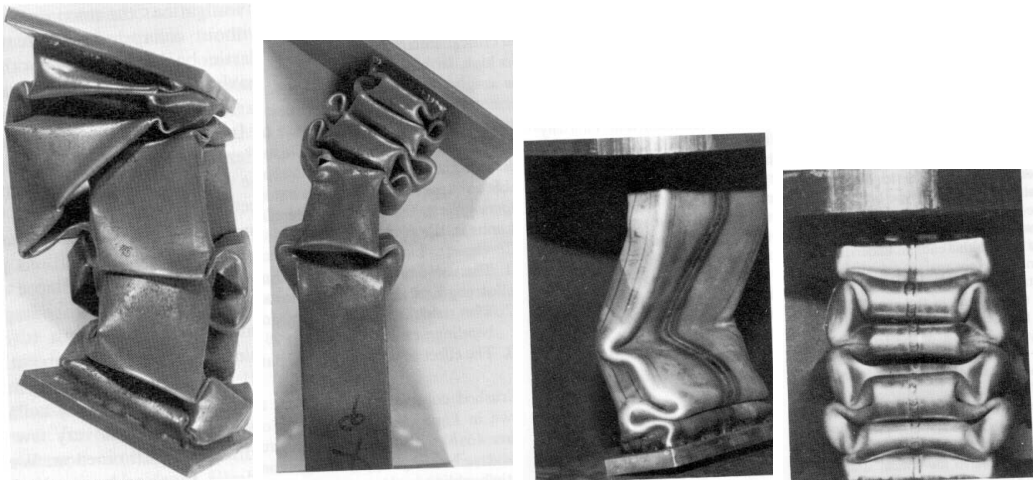

Figure 1.3: Different deformations on an axial loaded beam showing the wide variety of results for physical testing with an unstable model [1].

1.2 Related Research

Advancements in computational analysis has improved the design process of crumple zones however there are still limitations. The finite element method (FE method) does not replace the need for physical testing entirely, but it is an extremely useful tool to narrow down the different design options, leaving just a few for prototyping and physical verification. The FE method, in most cases, will have infeasible computational costs without the help of a design of experiments (DOE) technique. With the use of a meta-model and DOE technique, the number of samples can be dramatically decreased to only simulate parameters achieving certain results. For example, a DOE could be utilized

to only test material parameters that achieve the minimum energy absorption for the model. There is a lot of research covering this topic, and these techniques are widely used in industry today.

This technique has its limitations, it happens often that the parameters providing the optimal numerical results do not provide the desired deformation mode. In other cases, even with the desired deformation of the part in question, this may result in an undesired chain reaction to the rest of the assembly, causing other parts to fail. When this occurs it can be extremely time consuming to open the result files for each simulation result to inspect the geometrical deformations to verify success.

Rodrigo Iza Teran has developed a very useful solution to this problem. In his research he uses dimension reduction methods and spectral decomposition to provide a new way to post-process simulation bundle results [6]. The dataviewer tool, programmed by the Fraunhofer institute SCAI, implements his research into a tool for engineers. This tool loads the resulting geometries of the complete simulation bundle so one could quickly see if the desired deformation was achieved. In addition it uses spectral decomposition to classify different results by certain spectral coefficients. This can be used for clustering results and can help one conclude what parameters could be triggering different deformation modes.

Use of spectral decomposition for geometric classification isn't a widely researched topic. Spectral classification is used in image processing and is used on meshes for feature detection and database retrieval as discussed in []. The related research for use of spectral decomposition for engineering applications is even more limited. Although limited, the small amount of research in this area is extremely promising and could greatly impact the efficiency of engineering techniques used in industry today.

The research in this paper explores the use of a spectral decomposition in order to optimize the FE method with respect to specific deformation modes.

Chapter 2

Mathematical Concepts

The following section will propose the mathematical concepts used as the foundation for the following research. In order to do a spectral decomposition of a geometry one must first preform dimension reduction. The eigen-decomposition must be preformed on a square matrix where the size is equal to $[n \times n]$ where n is equal to the number of points on the surface (nodes). This matrix is a 'database' containing all the information about the relationships between two points. Dimension reduction methods are used to create the 'database' considered for spectral decomposition. There are many different dimension reduction methods however, the method for dimension reduction used in the following research is the Laplace-Beltrami method, or Laplician for short.

2.1 Laplace-Beltrami Operator

The Laplace-Beltrami operator is the divergence of the gradient [2],

$$\Delta f = div\ grad\ f, \tag{2.1}$$

The Laplace-Beltrami method depends on distances measured between points on a surface. This applies nicely to surface meshes because they are constructed by a set of points (nodes) that when connected represent a shape. By using this method effectively the shape, also known as a differential geometry, will be represented as a 2D manifold in R^3 space. There are a number of different schemes for the discrete Laplacian, the following section will introduce two of them. The first being the cotangent scheme and the second being the Belkin scheme.

The cotangent scheme is widely used in computer graphics, it is a discrete Laplace-Beltrami operator that focuses on the 1-ring neighborhood of a point. This means that the following equation is used to computed the discrete Laplacian for each set of adjacent nodes on a mesh. This discrete operator K is defined by [4]

$$K(x_i) = \frac{1}{2A_i} \sum_{x_j \epsilon N(x_i)} (cot\alpha_{ij} + cot\beta_{ij})(x_i - x_j), \tag{2.2}$$

Where A_i is the Voronoi region area of x_i, N_i is the single 1 ring neighborhood around point x_i. The line connecting points x_i and x_j is defined as an edge and α_{ij} and β_{ij} are angles opposite to the edge as seen in Figure 2.1.

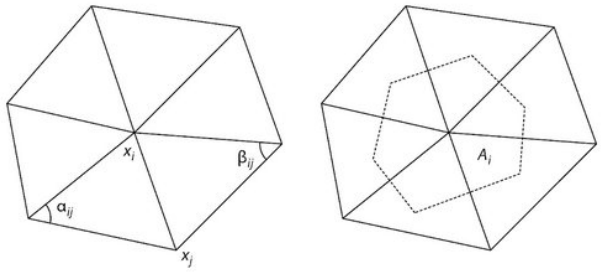

Figure 2.1: Discrete Laplace-Beltrami variables. The left image shows points x_i and x_j which are 1 ring neighbors and the two angles α_{ij} and β_{ij} that are opposite to the edge. The right image shows the Voronoi area of a point [2].

The Voronoi region is calculated by,

$$A_i = \frac{1}{8} \sum_{j \epsilon N(x_i)} (cot\alpha_{ij} + cot\beta_{ij}) \| \ x_i - x_j \ \|^2, \tag{2.3}$$

The cotangent scheme has shown to produce non-symmetric matrices resulting in complex eigenvectors, meaning it can't be used in applications depending on the eigenstructure for geometrical operations [5]. By using angles the cotangent scheme is in return heavily influenced by the geometry of the triangles, or the geometry of the mesh. Another limitation of the method is there isn't a clear method to extend point-point calculations beyond one neighborhood around the point.

The Belkin scheme proposes a mesh Laplacian operator L_M^h, where M is a mesh and h corresponds to the size of the neighborhood at a point. This means the operator is not constrained by only adjacent node pairs and is not influenced by the geometry of the elements. This means it works with faces made up of any number of vertices, and theoretically would also work with meshes containing mixed element types (i.e. tri, quad, or mixed elements).

$$L_M^h f(x) = \frac{1}{4\pi h^2} \sum_{t \epsilon M} \frac{Area(t)}{\#t} \sum_{y \epsilon V(t)} e^{-\frac{\|x-y\|^2}{4h}} (f(y) - f(x)), \tag{2.4}$$

where t is defined as a face on the mesh M and $\#t$ refers to the number of vertices on t. This scheme is independent of the geometry of the mesh, and considers all node pairs in the mesh. Node pairs with distances exceeding the size of the defined neighborhood are not considered in the approximation for this method. The method for approximation is defined in algorithm 1 and in appendix A.3.

Algorithm 1: Belkin Scheme

input : Vertices, V, with size $n \times 3$, where n = number of vertices.

Faces, F, with size $f \times p$, where f = number of faces and p = vertices per face.

output: Eigenvectors, $Evecs$

begin

foreach F **do**

  foreach $V \epsilon F$ **do**

    $A_V = \frac{Area(F)}{p}$

foreach V_i **do**

  foreach V_j **do**

    $d_{ij} = dist(V_i, V_j)$;

    if $d_{ij} > (\rho\sqrt{h})$ **then**

      $L(i,j) = (\frac{1}{4\pi h^2})(A_i)(A_j)(e^{-\frac{d_{ij}^2}{4h}})$

    else

      $L(i,j) = 0$

  $L(i,i) = -\sum L(i,:)$;

$Evecs = Eigs(L)$

The result of the Laplace-Beltrami operator is a square matrix that's size is equivalent to the number of nodes on the mesh. From the square matrix eigenvectors, λ, and eigenvalues, ψ, are calculated, which must satisfy the following equation,

$$\Delta_M \psi = -\lambda \psi, \tag{2.5}$$

where Δ_M is the Laplace-Beltrami operator on manifold M.

Due to the high nodal count in typical Finite Element Analysis (FEA) calculations, only a

subset of eigenvectors will be considered. This is justified due to related research showing that only a subset of eigenvectors are needed for reconstruction, Rodrigo Iza Teran proposes that accuracy of the reconstruction increases as the number of eigenvectors used for reconstruction increases. He also proposes that after a certain amount of eigenvectors, the increased accuracy for each additional eigenvector is not worth the added computational cost. He suggests 100 eigenfunctions for reconstruction as the optimal amount for the application studied in his thesis []. Due to the similarity of his application to the one in this paper, 100 eigenvectors will be considered, as well as 500 for comparison.

2.2 Spectral Coefficients

The following calculations and experiments are proven under the assumption that all deformations are isometric, meaning the transformation (or deformation) of the nodes on the mesh is distance preserving. By making this assumption it can be stated that for all timesteps the Laplacian matrix and eigen-decomposition will remain the same. Using this assumption one can calculate spectral coefficients at each timestep of the simulation. This creates a direct correlation between part deformation and coefficient magnitude.

Algorithm 2: Spectral Coefficients

input : Eigenvectors with length equal to node number, $Evecs$, Node coordinates for each simulation at each timestep, $coords_{ij}$, Timesteps, i, and Simulations, j

output: Spectral Coefficients, SC

begin

1 **foreach** i **do**

2 **foreach** j **do**

3 $SC_{ij} = coords_{ij} * Evecs$;

The eigenfunctions complete the basis for the Hilbert space $L^2(M)$, it is proposed in the

book 'Diffusion-Driven Wavelet Design for Shape Analysis' that for manifolds without boundaries and for manifolds with Neumann-conditioned boundaries, that the first eigenvalue is equal to zero, $\lambda_0 = 0$ and that the first eigenfunction is constant everywhere on the manifold,

$$\psi_0(x) = \frac{1}{\sqrt{\mu(M)}}. \tag{2.6}$$

This eliminates the need to set fixed values for eigenfunctions at the boundary [3].

Chapter 3

Workflow

The workflow consists of three main stages, pre-processing, simulation, and post-processing. The pre-processing stage consists of calculations from the undeformed mesh, then in the second stage, an optimization is preformed to obtain a simulation bundle. The majority of steps are in the final stage, post-processing.

Aside from the te simulation bundles calculated using LS-Opt, the other calculations are done using the MATLAB code documented in Appendix A.

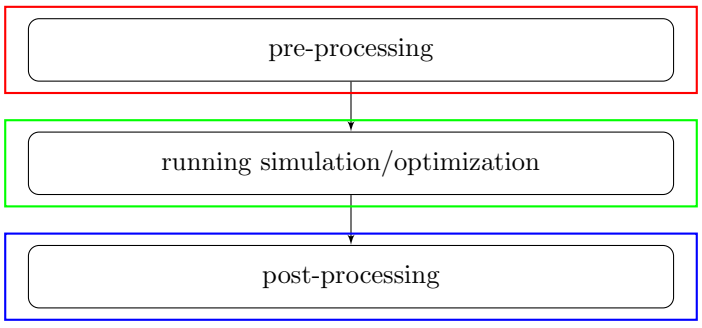

Figure 3.1: Workflow Overview

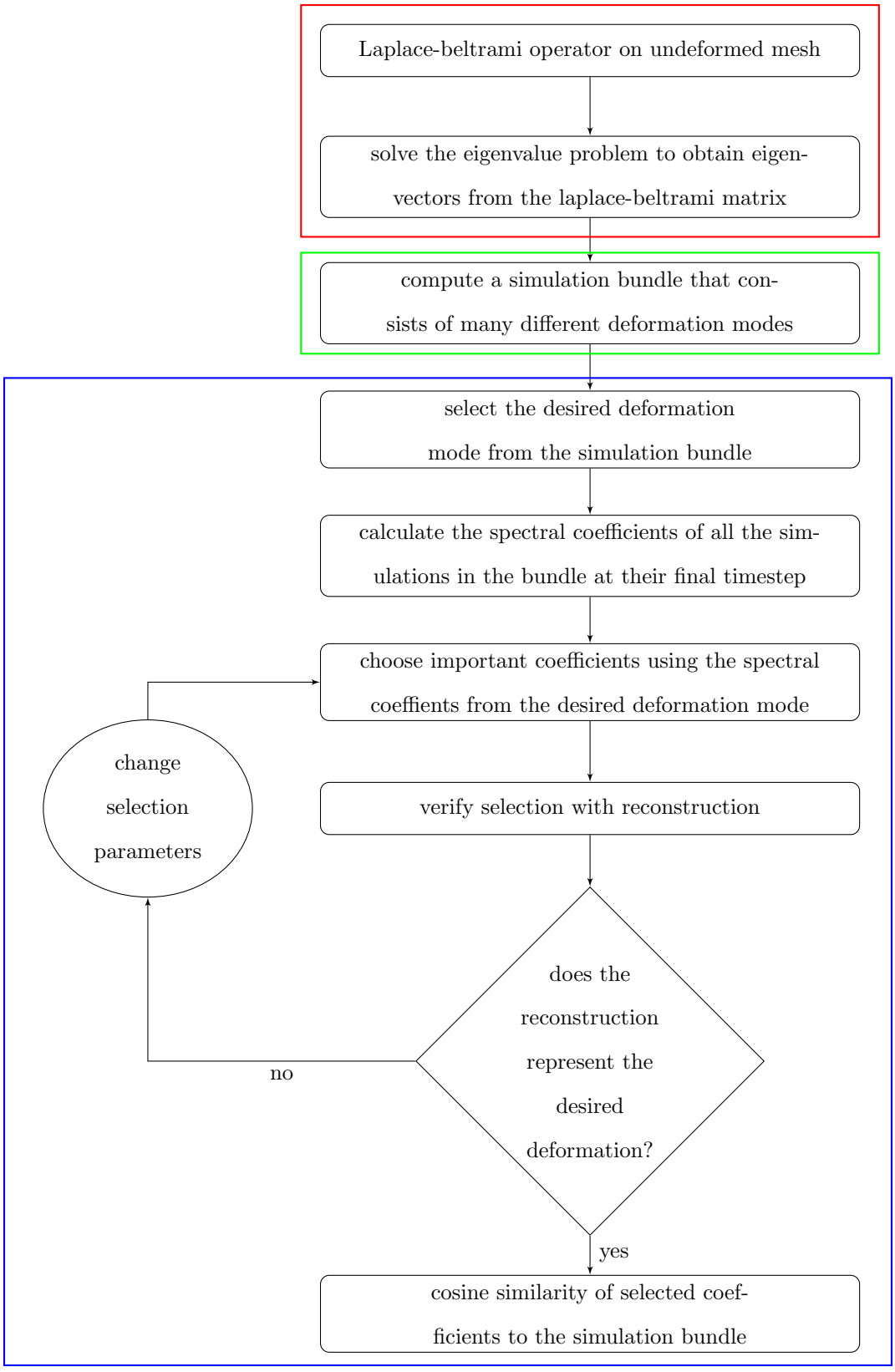

Figure 3.2: Current Workflow

3.1 Pre-processsing

in figure 3.2, the first two blocks are characterized as the pre-processing stage. This is because this work is done, or can be done before any simulations have been run. The laplace-beltrami operator, and the eigenvalue problem only need to be solved once for the given mesh. This is due to the assumption of isometric deformations.

The Laplace-beltrami operator is solved on the given mesh using the matlab code displayed in Appendix A.3.

The eigenvalue problem is then solved in the matlab code documented in Appendix A.5.

3.2 Simulation

This stage consists of block 3 in figure 3.2. The current workflow is optimized for a filtration from a database, this means that the simulation files making up the database must be obtained.

This experiment was designed to focus the part deformation at defined locations along the length of the part. In order to do this without changing the part geometry 'notches' were created from the elements on the flange of the hat section. The LS-opt was set up to change the thickness of the material at each notch location in order to simulate the removal of the material at that point. In addition the material thickness of the entire part was varied between 1 mm and 10 mm for each of the 10 notch locations. This resulted in a simulation bundle consisting of 100 simulations with a large variety of deformation modes.

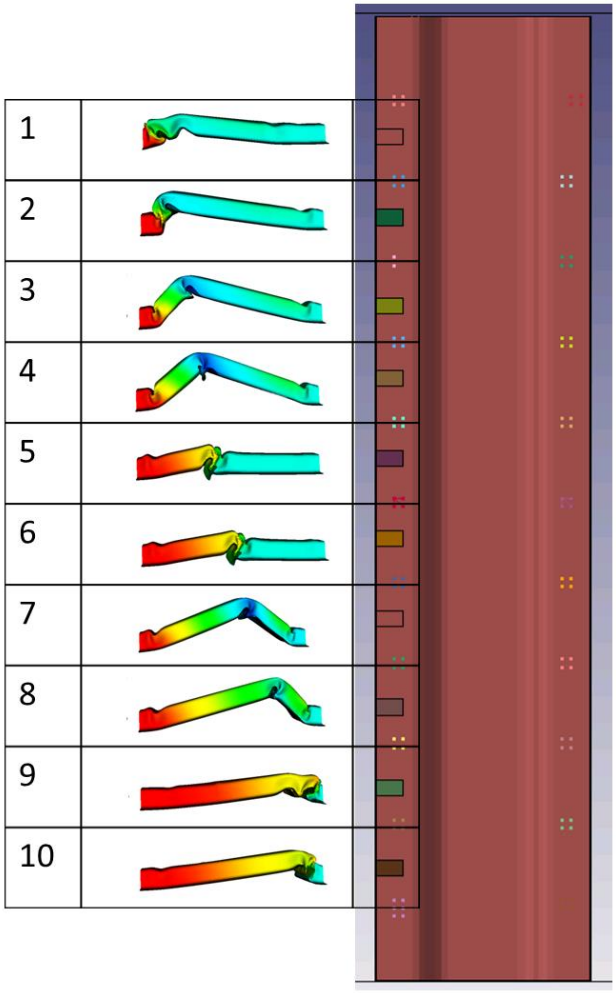

Figure 3.3: Deformation at the Notches when Thickness = 1mm.

3.3 Post-processing

The first step in the post-processing stage is to manually set a deformation target. This target corresponds to the ideal geometric deformation that the code will in turn use to return additional simulations with similar deformations. For this research application this step is done by visual inspection, this is done under the assumption that the engineer preforming the optimization knows how the part should be deforming with the giving test parameters defined in the simulation stage of this workflow.

The target deformation defined for this research is a part exhibiting an axial crush in the center of the beam, this ideal deformation is pictured below in figure 3.4

Figure 3.4: Part defined as the target for the remainder of the optimization, its deformation exhibits an axial crush deformation mode which the ideal deformation for the given test parameters.

The Cartesian coordinates for the nodes at each timestep of the simulations in the bundle are read from the d3plot files using the MATLAB code documented in Appendix A.4. These node locations are then used to calculate the spectral coefficients for all simulations in the bundle at each timestep. This spectral data is used to calculate the representative eigenvectors in the desired deformation mode.

3.3.1 Method for determining representative eigenvectors based on target deformation

The representative eigenvectors are determined by calculating which eigenvectors deviate from the mean in the X, Y and Z directions. This is done by first taking the mean and standard deviation of the X, Y, and Z coefficients respectively. The important coefficients are then defined as the coefficients that are 2 standard deviations above the mean coefficient value.

The eigenvectors are then scaled by the by the important coefficients, the sum of these scaled eigenvectors is used for reconstruction.

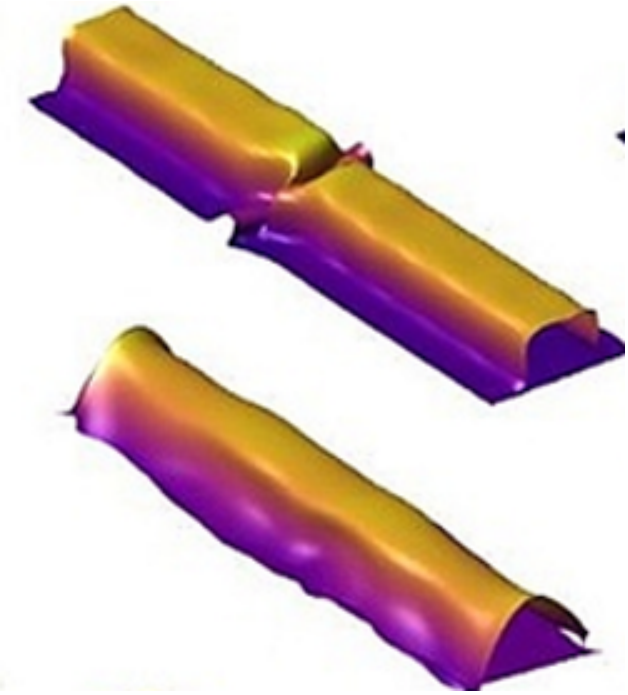

Figure 3.5: The part on the top is a reconstruction of the axial deformation mode using the first M = 500 coefficients ranked by the magnitude of their corresponding eigenvector, it is plotted for easy comparison to the below reconstruction which is done using M = 14 coefficients selected by the method described above and documented in Appendix A.8.

Based on the reconstruction pictured in Figure 3.5 the selected M = 14 important coefficients are determined effective for a low dimensional representation of the target deformation mode.

If this was not determined an effective representative this is were the iteration step pictured in Figure 3.2 would occur. The coefficients could be adjusted by selecting everything 1 standard deviation above the mean, or all values above the mean. If the number of coefficients, M, are too high than the value could be adjusted to be 3 standard deviations above the mean or higher. For the purpose of this research, 2 standard deviations have been determined sufficient for representation of this deformation.

3.3.2 Database filtration using cosine similarity

After visual verification, the same important coefficients will be filtered out of all simulations in the bundle and the same reconstructive eigenvector is calculated. Using the reconstruction vector for the target deformation mode, cosine similarity is calculated between it an every other simulation in the bundle. This step is documented in Appendix A.9.

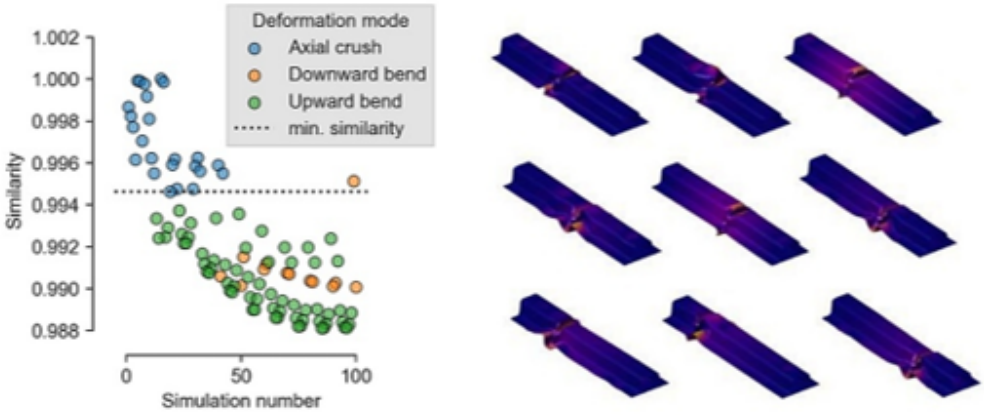

Figure 3.6: Pictured on the left is the cosine similarity of all deformations in the bundle using the descriptor for the axial deformation mode, the points on the plot have been colored to indicate which simulations in the bundle are upward bend, downward bend or axial deformations. On the right the 9 most similar simulations to the axial deformation mode have been pulled and plotted from the simulation bundle for further verification.

3.4 Evaluation

Based on the verification above the workflow defined is effective in determining important spectral coefficients as they pertain to a specific deformation mode. These coefficients are then used to do a low dimensional reconstruction of the part. Further analysis allows one to use these coefficients as a target for a database filtration.

Chapter 4

Results

Using the spectral coefficients for a low dimensional reconstruction isn't novel, however traditionally the reconstruction is done using the first M eigenvectors based on the magnitude of their corresponding eigenvalue. This is effective in construction high frequency geometric features however details are compromised in this method. The fine details are important when considering specific deformation modes. In order to evaluate the results of the workflow outlined in Chapter 3, reconstruction method for all 3 deformation modes are used for comparison.

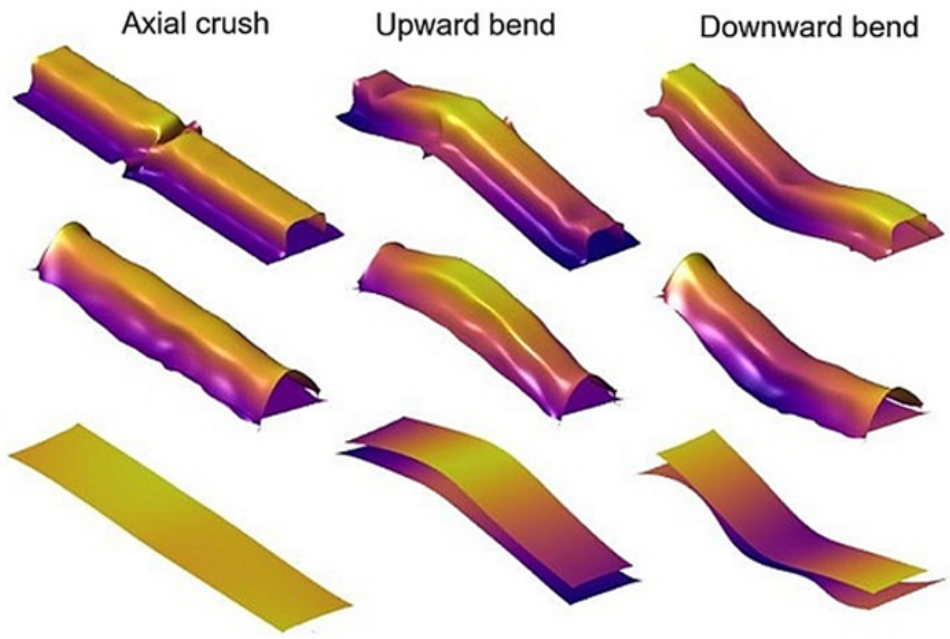

Figure 4.1: The top shows a reconstruction using M = 500 eigenvectors based on the magnitude of their corresponding eigenvalue. The middle row represents reconstruction from the proposed descriptor with sizes M = 14 for axial crush, M = 17 for upward bend, M = 16 for downward bend. The bottom row shows a reconstruction using M = 14 for axial crush, M = 17 for upward bend, M = 16 for downward bend however these eigenvectors are sorted using the traditional method rather than the proposed descriptor.

The Figure 4.1 above shows the benefit of the workflow outlined in chapter 3 when compared to traditional low dimensional reconstruction methods. The results of this research confirm the ability to use this method of finding a spectral descriptor specific to a defined target, for more detailed low dimensional reconstruction and for effective filtration from a database.

The figures below, Figure 4.2, Figure 4.3, and Figure 4.4 show the cosine similarity results when applying the spectral descriptors M = 14 for axial, M = 17 for upward bend, and M = 16 for downward bend. These results verify the descriptors ability to successfully rank simulations based on their correspondence to the specified deformation mode. The simulations closest to the X axis are the simulations that have the highest similarity to the descriptor.

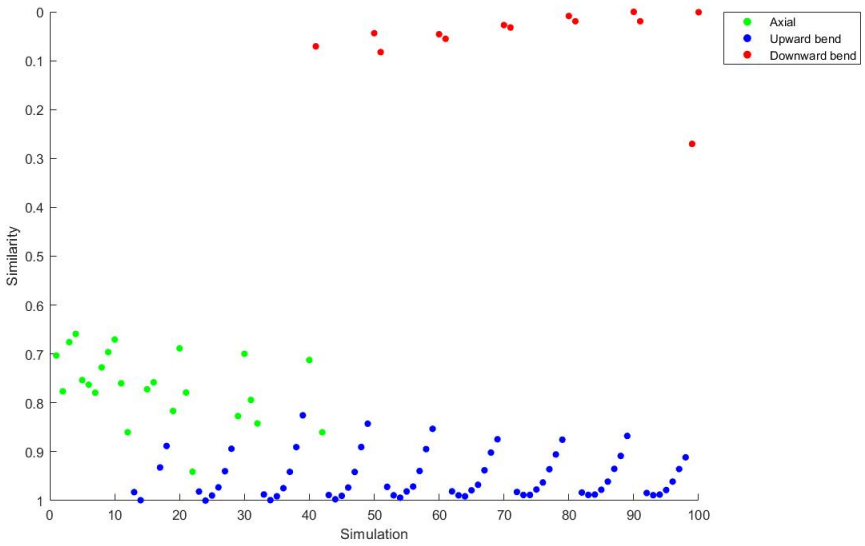

Figure 4.2: Normalized cosine similarity plot for upward bend deformation mode. X axis is the simulation, and y axis is similarity where 1 is equal to 100 percent similarity. Points closest to the X axis represent simulations with highest similarity to the deformation mode.

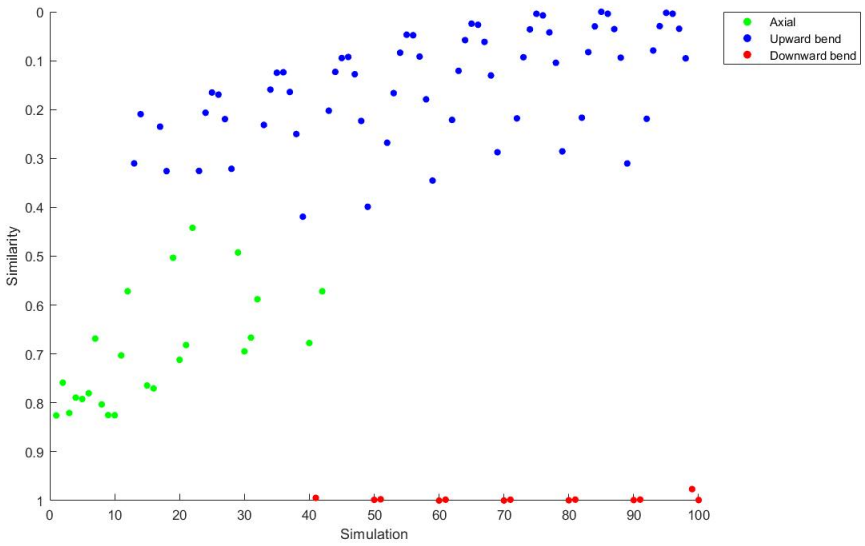

Figure 4.3: Normalized cosine similarity plot for downward bend deformation mode. X axis is the simulation, and y axis is similarity where 1 is equal to 100 percent similarity. Points closest to the X axis represent simulations with highest similarity to the deformation mode.

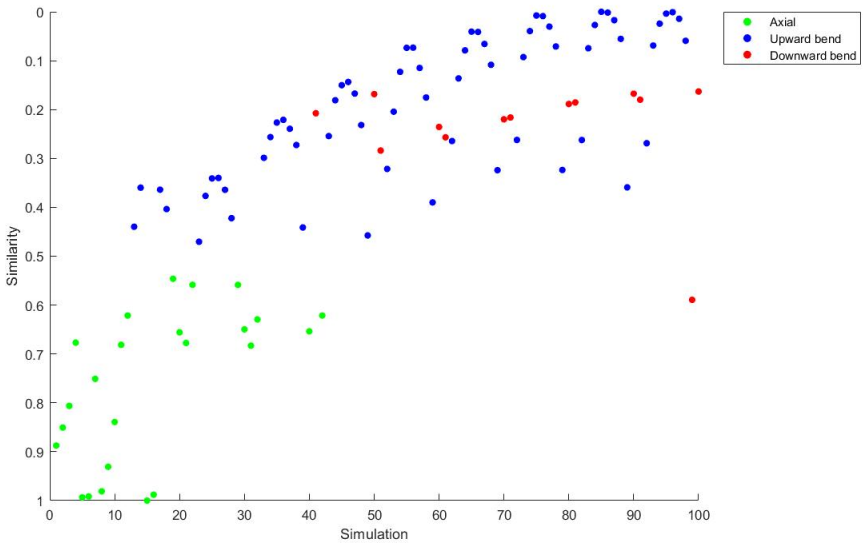

Figure 4.4: Normalized cosine similarity plot for axial deformation mode. X axis is the simulation, and y axis is similarity where 1 is equal to 100 percent similarity. Points closest to the X axis represent simulations with highest similarity to the deformation mode.

In Figure 4.2 the blue dots, or upward simulations, show curvature withing the different notch increments. For example, simulations 1-10 correspond to a specific notch with thicknesses increasing by 1mm in each simulation. These curves indicate that for most of the notch locations, part thickness equal to 4mm results in an upward bend. This information would then be valuable to an engineer wishing to obtain an upward bend deformation mode.

The next set of figures plotted below are binary plots of the coefficients for the representative deformation modes. The Y axis is the coefficients 1-500 and they are ordered by the magnitude of their corresponding eigenvalue. The X axis is time-steps of the simulation 0-20. The purpose of these plots is to see if coefficients in the higher frequency range are above the mean for specific deformation modes. Also the time-step was plotted to see if certain coefficients passed the mean threshold as the part continued to deform, in addition to coefficients dropping below the threshold as the part deformation progresses.

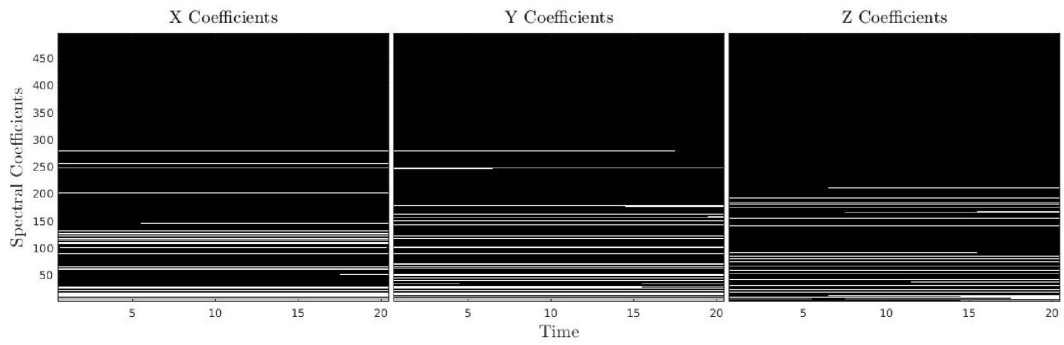

Figure 4.5: The y-axis is the spectral coefficients ordered by the magnitude of their corresponding eigenvalue, the x-axis is the time-steps 1-20 of the model part for the upwards bending deformation mode. All coefficients above the mean are plotted in white and all others in black. The coefficients above the mean are considered the driving coefficients for this specific deformation mode, however in this research only coefficients 2 standard deviations above the mean are part of the descriptor.

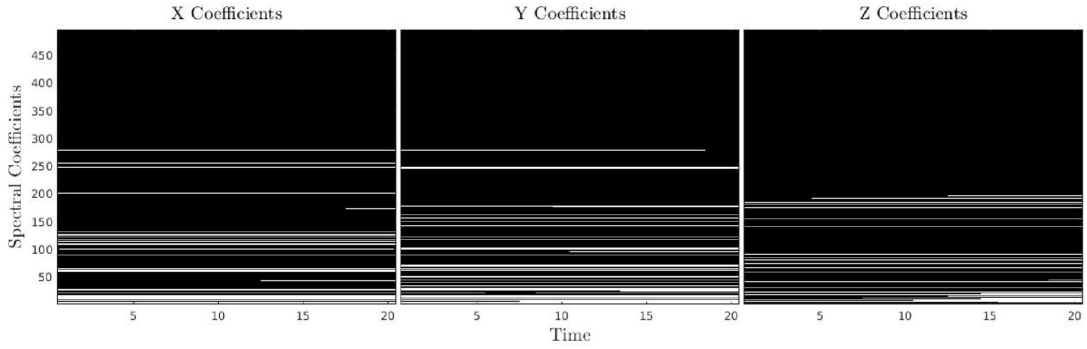

Figure 4.6: The y-axis is the spectral coefficients ordered by the magnitude of their corresponding eigenvalue, the x-axis is the time-steps 1-20 of the model part for the downwards bending deformation mode. All coefficients above the mean are plotted in white and all others in black. The coefficients above the mean are considered the driving coefficients for this specific deformation mode, however in this research only coefficients 2 standard deviations above the mean are part of the descriptor.

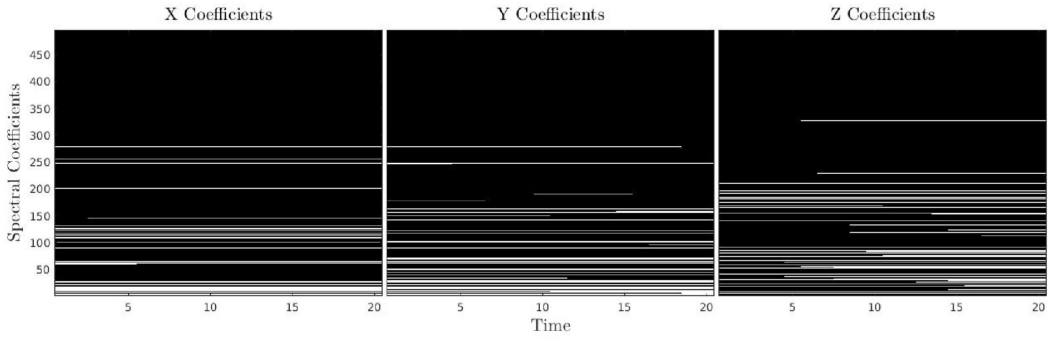

Figure 4.7: The y-axis is the spectral coefficients ordered by the magnitude of their corresponding eigenvalue, the x-axis is the time-steps 1-20 of the model part for the axial deformation mode. All coefficients above the mean are plotted in white and all others in black. The coefficients above the mean are considered the driving coefficients for this specific deformation mode, however in this research only coefficients 2 standard deviations above the mean are part of the descriptor.

The binary plots confirm the assumption that certain coefficients become either more prominent as deformation progresses or less prominent. In addition the figures above, Figure 4.5, Figure 4.6, and Figure 4.7 show that certain high frequency coefficients are prominent on specific modes. This verifies that the coefficients to best represent the deformation mode are not necessarily correlated to the magnitude of the eigenvalue, rather than their magnitude in comparison to the other coefficients. The coefficients deviating the furthest from the mean indicate high importance in the geometry of the designated deformation mode.

Chapter 5

Conclusion

Advancements in computational simulation have reduced costly and time consuming physical crash testing. Currently simulation is still limited by certain things. There is currently no way to define a complex deformation mode in a FEA analysis. There are methods engineers use to track individual node displacements however complicated deformation modes such as folding or twisting cannot be easily tracked by individual nodes.

The purpose of this research is to develop a method that can be used to sort through simulation files to find all simulation results with the desired deformation mode. This is important for automotive engineering applications because in addition to other parameters in crash testing, physical deformation should be preserved in order to optimize the crashworthiness of the vehicle.

5.1 Contributions

The research in this paper has contributed to te scientific community by presenting a novel way to use spectral descriptors to define a deformation mode in order to do low

dimensional reconstructions and database filtration.

5.2 Additional applications

This method could also be applied in computer graphics in order to make visually accurate low dimensional reconstructions. In the engineering domain there are endless applications in finite element simulation and part optimization. Another interesting application of this research would be to preserve different deformation modes when calculating a topology optimization.

5.3 Future work

The future work on the research project would be to incorporate this method into a simulation optimization. This descriptor would be used as parameter in a machine learning optimization approach. Another study that would further validate this method would include a robustness study with different part geometries and deformation modes.

5.4 Summary

To summarize the research above, a method was developed to used spectral analysis of part deformations in order to come up with a spectral descriptor that represents a complex deformation mode. The importance of this spectral descriptor is to be able to define a deformation mode during a design optimization, or filter through a simulation bundle to find results matching the target deformation mode.

Bibliography

[1] Paul Du Bois, Clifford C. Chou, Bahig B. Fileta, Tawfik B. Khalil, Albert I. King, Hikmat F. Mahmood, Harold J. Mertz, Jac Wismans, Priya Prasad, and Jamel E. Belwafa. Vehicle crashworthiness and occupant protection.

[2] Tingbo Hou and Hong Qin. *Diffusion-driven Wavelet Design for Shape Analysis*. AK Peters/CRC Press, 2014.

[3] Hesham Kamel Ibrahim. *Design optimization of vehicle structures for crashworthiness improvement*. Citeseer, 2009.

[4] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. 2001.

[5] Fabiano Petronetto, Afonso Paiva, Elias S Helou, DE Stewart, and Luis Gustavo Nonato. Mesh-free discrete laplace–beltrami operator. In *Computer Graphics Forum*, volume 32, pages 214–226. Wiley Online Library, 2013.

[6] Rodrigo Iza Teran and Jochen Garcke. Geometrical methods for the analysis of simulation bundles. *Doktorarbeit, Universität Bonn, eingereicht*, 2016.

Appendix A

START.m

```
%   Skylar Sible HRI-EU 14.12.2018

clc
clear

%START HERE Update appropriate values

% Fuctions writen by Skylar Sible at HRI-EU
addpath('functions')

addpath('functions/functionsRough')

% Fuction written by others copied from open source websites
addpath('functions/Borrowed Functions')


load('C:\Users\Skylar\SkyDrive\Documents\Research\mesh_data.mat')
 %%  Insert appropriate values below: All these values should be updated prior to running the code
%
% %   Path to dataviewer files, see examples below
%
% %       Hat section and Bottom plate with notches included
%         path = '/hri/localdisk/ssible/dataviewerprojects/BeamCrush.100samples.withnotch/file/';
%
% %       Hat section alone, with notches included (this would result in IP =
% %       1, see below for details
% %       path = '/hri/localdisk/ssible/dataviewerprojects/BeamCrush.100samples.withnotch/file2/';
% %       notches only hat section
%
% %       Hat section and Bottom plate without the notches
% %       path = '/hri/localdisk/ssible/dataviewerprojects/BeamCrush.Test6.Test6/file/';
% %       without notches
%
%
```

```
35 %
36 %% Path to the keyword file, see example below
37 %
38 %%       This file determines the geometry of the part/parts, it can be
39 %%       found in the folder for the simulation bundle.
40 %
41 %%       This location only needs to be updated if the part chages, however
42 %%       if you want to analyze a different simulation bundle of the same
43 %%       geometry it is not nessesary to update. However for consistency it
44 %%       is recomended.
45 %
46 %        keyword_path = '/hri/localdisk/ssible/LSfiles/BeamCrush/Test7/Test7.k';
47 %
48 %% Path to the simulation bundle, see example below
49 %
50 %%       sim_bundle_file_path =
51 %%       '/hri/localdisk/ssible/LSfiles/BeamCrush/Test6/Test6/Stage1/';%
52 %%       without notches
53 %        sim_bundle_file_path = ...
54 %            '/hri/localdisk/ssible/LSfiles/BeamCrush/Test7/Stage2/';
55 %%           with notches
56
57 %%
58
59 %   Number of timesteps and simulations in the bundle
60
61 %   Timesteps go from 1 to the specified number below, this number can be
62 %   less than the total number of timesteps but not greater.
63     timesteps = 20;
64
65 %   This corresponds to the number of simulations in the bundle. It will
66 %   evaluate simulation 1 to the number specified. If you choose to
67 %   evaluate only specific simulations this can be determined below.
68     simulations = 100;
69
70 %   IP stands for independent parts, this means the number of disconected
71 %   parts in your analysis This value is used later to filter out the first
72 %   eigenvectors from the analysis, if there ate 2 parts then the program
73 %   would filter out the first 4 eigenvectors
74     IP = 0;
75
76 %   2 eigenvectors for each disconected mesh
77     IP = 2*IP;
78
79 %       There is a potential that this number could be greater or less that
80 %       2 eigenvectors per independent part, there is a potential that this
81 %       value could increase or decrease depending on how symmetric the
82 %       part is. This value could also be fixed at 2 if it remains true
83 %       that the first value corresponds to center of mass and the second
84 %       corresponds to the distribution of mass, from the center along the
85 %       axis with the highest dribution of mass.
86
87
88 %% Function reads dataviewer files in specified path
89 %   It returns eigenvalues, eigenvectors, and all spectral coefficients It
90 %   only returns the highest 100 eigenvalues and their corresponding
```

```
91  %   eigenvectors It also returns the nodeids, these ar important in order
92  %   to reorder the eigenvectors and to only include the parts specified in
93  %   the dataviewer calculation.
94
95  %   If you choose to skip the dataviewer completely, make sure to import an
96  %   vector of the node ids corresponding to the specific parts you would
97  %   like to include.
98
99  %     [Evals, Evecs, nodeids] = read_dataviewer(path);
100 %
101 % %   Sort node ids into correct order
102 %     [nodeids nodeids_order] = sort(nodeids, 'ascend');
103 %
104 % %   Sort eigenvectors so that the value in the vector matches the order of
105 % %   the nodes
106 %     Evecs1 = Evecs(nodeids_order,(IP+1):end);
107 %
108 % %   Sets the proper variables for the Eigen vectors to be used throughout
109 % %   the rest of the code
110 %     Evecsall = Evecs(nodeids_order,:);
111 %     Evecs = Evecs1;
112
113 %%  Calculate the mesh geometry
114 %   This reads out the node coordinates of nodes specified in nodeids It
115 %   also calculates a triangulation of the nodes to determine the nodes
116 %   belonging to each face
117
118 %     [ mesh ] = mesh_geometry(keyword_path, nodeids);
119
120
121 %% My own laplace-beltrami calculation
122 %   Input is the mesh variable calculated above Second Input is the number
123 %   of eigenvectors you would like to calculate
124
125     [LB] = Laplace_Beltrami_Operator(mesh, 500);
126
127 %   Sets the proper variables for the Eigen vectors to be used throughout
128 %   the rest of the code
129
130     Evecs = LB.evecs(:,(IP+1):end);
131     Evecsall = LB.evecs;
132
133   %%  Read node displacement coordinates for each simulation at each timestep
134
135    % [node_displacement] = d3plot_read(sim_bundle_file_path, timesteps,...
136    %    simulations, nodeids);
137
138     [Spectral_Coefficients_Calc] = Spec_coeff(node_displacement, Evecs);
139
140 %%  Create clusters of different deformation modes and calculate a new
141 %   arbiturary mesh representing one cluster
142
143 %   Define the number of clusters you want to create
144 %     modenum = 3;
145 %
146 %    %spectral clustering = 'S'
```

```
147 %    [Deformation_modesS, mode_simsS] = deformation_modes('S',...
148 %        node_displacement, modenum, mesh, Spectral_Coefficients_Calc);
149 %
150 %    %geometric clustering = 'G'
151 %    [Deformation_modesG, mode_simsG] = deformation_modes('G',...
152 %        node_displacement, modenum, mesh, Spectral_Coefficients_Calc);
153
154 %% Choose the best cluster direction and method for your individual part
155
156 %     Deformation_modes = Deformation_modesS.Z;
157 %     mode_sims = mode_simsS.Z;
158 %
159 %% Set representative deformations as the 'modes'
160
161 %  5- axial, 24- upward, 60- downward
162
163     Deformation_modes = node_displacement([5,24,60],:);
164     mode_sims = {5,24,60};
165
166 %% Calculates the spectral coefficients based on the new geomery of
167 %  the "deformation modes"
168
169     [Spectral_Coefficients_New] = Spec_coeff(Deformation_modes, Evecs);
170     [Spectral_Coefficients_All] = Spec_coeff(Deformation_modes, Evecsall);
171
172 %%      Scale eigenvectors based on spectral coefficients of the final
173 %       timestep of specified simulation then plot....
174
175     desired_deformation_mode =  1;
176
177     [coeff] = time_frequency_cluster_diff(Spectral_Coefficients_Calc,...
178         Spectral_Coefficients_New, desired_deformation_mode, mode_sims,...
179         'C');
180
181 %   important eigenvectors
182     [Evecmode, Evecs_scaled, Eval_important] = ...
183         important_eigenvectors(coeff, Evecs, Spectral_Coefficients_New,...
184         desired_deformation_mode, 2);
185
186 %   plot
187     new_eigenvector_on_mesh(Evecmode, mesh, desired_deformation_mode);
188
189 %   Calculating the similarity
190
191     A = Spectral_Coefficients_New{desired_deformation_mode, timesteps};
192
193     [ similar ] = similarity(node_displacement, A, Eval_important, Evecs);
194
195 %   Reconstruction Comparison with Important eigenvectors
196     [total_unique_eigenvectors, unique_eigenvectors, AB2, AC2] = ...
197         reconstruction_comparison(Spectral_Coefficients_All,...
198         desired_deformation_mode, IP, mesh, Evecsall, Eval_important);
199
200 %   Save Eval_important as Eval_important#, # = mode number
201     Eval_important1 = Eval_important;
202
```

```
%%

    deformation_mode_numbers = [1 2 3];
    Eval_importantm = [Eval_important1, Eval_important2, Eval_important3];

    reconstruction_comparison_all(Spectral_Coefficients_All,...
        deformation_mode_numbers, IP, mesh, Evecsall, Eval_importantm);

%%  Plots all simulations specified, in this case all simulations
%   acheiving the desired deformation mode


    [plot] = plot_final_timestep(similar(1:10), mesh, node_displacement);


%%  Saving Parameters above to be used to filter future simulations

%  command = ['mkdir',' ',sim_bundle_file_path, 'Desired_Deformation'];
%  status = system(command);
%
%
%
%
%  Desired_Deformation.Evecs = Evecs; Desired_Deformation.A = A;
%  Desired_Deformation.nodeids = nodeids;
%  Desired_Deformation.Eval_importantX = Eval_importantX;
%  Desired_Deformation.Eval_importantY = Eval_importantY;
%  Desired_Deformation.Eval_importantZ = Eval_importantZ;
%
%
%  save([sim_bundle_file_path,'Desired_Deformation/',...
%   'Desired_Deformation.mat'],'Desired_Deformation');

%%

time = (1:20);

E = (1:6);

view = 2;

eigenvectors_as_nodes_plot(E, Spectral_Coefficients_All,...
    desired_deformation_mode, Evecsall, mesh, time, view);

%%
```

A.1 read_dataviewer.m

```
function [ Evals, Evecs, nodeids ] = read_dataviewer(path)

%   read_dataviewer, Reads the file in the path specified and imports the
%   eigenvectors and eigenvalues so they can be used in calculations

```

```
%   INPUT:  path: is the path that leads to all the dataviewer files
%               Example:
%               /hri/localdisk/ssible/matlab/read_dataviewer/DataviewerAnalysisPratice2/

%   OUTPUT: Evals: Eigen values calculated by the dataviewer software
%           Evecs: Eigenvectors calculated by the dataviewer software

%% Gets eigenvalues and eigenvectors

%   Read eigen values

    Evalpath = [path, 'results_eigen_vectors/eigenvalues.txt'];
    Evals = load(Evalpath);

%   Read eigenvectors

    Evecpath = [path,'/results_eigen_vectors/eigenvectors.txt'];
    Evecs = load(Evecpath);

%   Reads node ids

    nodeIDpath = [path, 'results_eigen_vectors/nodeids.txt'];
    nodeids = load(nodeIDpath);


end
```

A.2 mesh_geometry.m

```
function [ mesh ] = mesh_geometry(keyword_path, nodeids)
%   mesh_geometry: finds geometry of keyword file by specified path
%   INPUT:  keyword_path: Path to the keyword file of the simulation
%       example:'/hri/localdisk/ssible/LSfiles/BeamTest3/BeamTest3.k'
%           nodeids: vector of the nodes included in the analysis, all
%           nodes belonging to the parts being analyzed
%   OUTPUT: mesh: struct of the geometry values containing,
%               mesh.geometry.TRIV: Faces
%               mesh.geometry.X: X coordinates
%               mesh.geometry.Y: Y coordinates
%               mesh.geometry.Z: Z coordinates
%               mesh.geometry.nodenum: number of nodes


%%   Read node coordinates for mesh keyword file

    meshk = fileread(keyword_path);
    meshk = textscan(meshk,'%s','Delimiter','\n')';
    meshk = meshk{1};


    nodename = strfind(meshk, '*NODE');
    line  = find(not(cellfun('isempty', nodename)));
    docend = strfind(meshk, '*END');
```

```
25      endline  = find(not(cellfun('isempty', docend)));
26
27      nodecoords = meshk(line+2:endline-1);
28
29      nodecoords = cell2mat(cellfun(@str2num,nodecoords,'UniformOutput',false));
30
31
32      elementname = strfind(meshk, '*ELEMENT_SHELL');
33      elementline = find(not(cellfun('isempty', elementname)));
34      elementend = strfind(meshk, '*NODE');
35      elementendline  = find(not(cellfun('isempty', elementend)));
36
37      elementids = meshk(elementline+2:elementendline-1);
38
39      elementids = cell2mat(cellfun(@str2num,elementids,'UniformOutput',false));
40
41
42      element_ids = elementids(:,3:6);
43
44      nodecoordinates = nodecoords(nodeids,2:4)';
45      nodecoordinates_with_ids = nodecoords(nodeids,1:4);
46
47
48      elms = ismember(element_ids,nodecoordinates_with_ids(:,1));
49
50      for i = 1:length(elms(:,4));
51
52      if elms(i,1) == 0
53          element_ids(i,:) = 0;
54      end
55      end
56
57      element_ids(all(element_ids == 0,2),:) = [];
58
59
60      %% Making the mesh triangular by splitting the quadralatteral elements
61
62
63      faces = [element_ids(:,1:3);element_ids(:,[1,3,4])];
64
65
66      for i = 1:length(nodecoordinates_with_ids(:,1));
67          for k = 1:3;
68
69              facesswitch = find(faces(:,k) == nodecoordinates_with_ids(i,1));
70              faces(facesswitch,k) = i;
71
72          end
73      end
74
75  %%
76
77      vertices = nodecoordinates';
78
79      faces2 = triangulation(faces,vertices(:,1),vertices(:,2),vertices(:,3));
80
```

```
81      faces = faces2.ConnectivityList;
82
83      vertices = faces2.Points;
84
85
86  mesh.geometry.TRIV = (faces);
87  mesh.geometry.X = vertices(:,1);
88  mesh.geometry.Y = vertices(:,2);
89  mesh.geometry.Z = vertices(:,3);
90  mesh.geometry.nodenum = length(vertices);
91
92  end
```

A.3 Laplace_Beltrami_Operator.m

```
1   function [LB] = Laplace_Beltrami_Operator(mesh, n)
2
3   % Laplace_Beltrami_Operator: does a laplace beltrami dimension reduction
4   % calculation on a triangular mesh
5   %   INPUT: mesh: struct containg the geometry information if the form
6   %   specified below
7   %           mesh.geometry.TRIV: Faces mesh.geometry.X: X coordinates
8   %           mesh.geometry.Y: Y coordinates mesh.geometry.Z: Z coordinates
9   %           mesh.geometry.nodenum: number of nodes n: number of requested
10  %           eigenvectors and eigenvalues to be output
11  %       OUTPUT: LB: struct containing of the folowing data
12  %                   LB.evals: eigenvalues LB.evecs: eigenvectors
13
14  %%  Import mesh
15
16  nodenum = mesh.geometry.nodenum;
17  faces = mesh.geometry.TRIV;
18  NC = [mesh.geometry.X mesh.geometry.Y mesh.geometry.Z];
19
20
21  %%
22  nodepairs = [faces(:,[1,2]);faces(:,[2,3]);faces(:,[3,1])];
23  nodepairs(nodepairs(:,1)>= nodepairs(:,2),:) = [];
24
25
26  node1 = nodepairs(:,1);
27  node2 = nodepairs(:,2);
28
29  %%  Calculating the weights
30
31  %   Distance between all points
32      dist = pdist(NC);
33      dist = squareform(dist);
34
35  %   The weight matrix
36      weight = zeros(length(node1),1);
37      for i = 1:length(node1)
38          weight(i) = dist(node1(i),node2(i));
```

```
39      end
40
41
42  %%  Geodesic distances considering weights
43
44      G = graph(node1,node2,weight);
45      geodist = distances(G);
46
47  %%  Determing the size of the neighborhood of a point
48
49  %       still not entirely sure on how to determing p and h, however these
50  %       values seem to work
51            h = 5;
52            p = 6;
53
54  %       These equations are from, Discrete Laplace operator on meshed
55  %       surfaces, by M.Belkin, J. Sun, and Y. Wang
56          ph = p*sqrt(h);
57          geodist(geodist(:,:) > (ph)) = 0;
58
59  %       vertexAreas is a function found online but its source is currently
60  %       unknown, although there is nothing arbitruary about it.
61          A = vertexAreas(NC, faces);
62          A = full(A);
63
64  %%
65
66
67  %   Square matrix [nodenum X nodenum]
68      L = zeros(nodenum);
69
70  %   loops throught all columns
71      for i = 1:nodenum
72
73  %       Loops through all rows
74          for j = 1:nodenum
75
76  %           Checks if the distance between the two nodes is within the
77  %           defined neighborhood of the point
78              if geodist(i,j)> 0
79
80  %               Sets the goedesic distance between the two points equal to
81  %               dij
82                  dij = geodist(i,j);
83
84  %               Area of each vertex, equal to 1/3 of the sum of all
85  %               surrounding faces
86                  Ai = A(j,j);
87                  Aj = A(i,i);
88
89  %               The belkin scheme equation for the laplace beltrami
90  %               operator at the pair of points
91                  Aij = (Ai*Aj*(exp(-((dij)^2)/(4*(h)))))/(4*pi*(h^2));
92
93  %               Setting the place in the laplace matrix equal to the result
94                  L(i,j) =  Aij;
```

```
95
96              end
97          end
98      end
99
100 %   The diagonal of the matrix is equal to sum of all of the values in its
101 %   row
102     Ldiag = sum(L);
103
104 %   Turning the vector into a diagonal matrix
105     Ldiag = diag(Ldiag);
106
107 %   Subtract the diagonal from the laplacian matrix
108     L = L - Ldiag;
109
110 %   clear Ldiag to save space/memory
111     clear Ldiag
112
113 %%  Eigen decomposition of the laplace-beltrami matrix
114
115 %   n defines the number of eigenvalues/vectors calculated
116     [Evecs2 Evals2] = eigs(sparse(L),n,-1e-6);
117
118 %   Store results
119     LB.evals = diag(Evals2);
120     LB.evecs = Evecs2;
121
122
123 end
```

A.4 d3plot_read.m

```
1  function [node_displacement] = d3plot_read(sim_bundle_file_path, timesteps, simulations, nodeids)
2  %%  Function creates folder and saves the node coordinates of each simulation at each timestep,
3  %   then reads these files and makes a cell matrices of all the
4  %   coordinates, the result is node_displacement, with rows being the
5  %   simulations and colunms being the timesteps
6  %
7  %   INPUT:
8  %       sim_bundle_file_path: see example below sim_bundle_file_path =
9  %       '/hri/localdisk/ssible/LSfiles/BeamCrush/Test7/Stage2/'; timesteps:
10 %       number of timesteps being analyzed simulations: number of
11 %       simulations being analyzed nodeids: ids of the nodes corresponding
12 %       to the parts being loaded, if you want to load all parts then
13 %       nodeids = [1:nodenumber];
14 %
15 %   OUTPUT:
16 %       node_displacement: cell matrix, rows = simulation number, columns =
17 %       timestep
18
19
20 %%  Read d3plot files
21
```

```
%   Checks if the file already exists, if it does it loads the file, if not
%   it executes the following code

if ~exist([sim_bundle_file_path,'node_displacement.mat'],'file');

%   Checks if the d3plot file has been read before, if not it executes the
%   following code
    if ~exist([sim_bundle_file_path,'Node_out/'],'file');

    %   Create a command file that will be executed by LS pre post in a
    %   later step
        commandfile_name = 'displaced_nodes.cfile';
        commandfile = [sim_bundle_file_path, commandfile_name];
        fid = fopen(commandfile, 'wt');

%
        command = ['mkdir',' ',sim_bundle_file_path, 'Node_out'];
        status = system(command);

    %       writes commands for each simulation specified
            for i = 1:simulations;

                fid = fopen(commandfile, 'wt');

    %           command to open the d3plot file in LS-prepost
                d3plot = [sim_bundle_file_path, '1.',mat2str(i), '/d3plot'];
                opencommand = ['open d3plot "',d3plot,'"'];
                fprintf(fid, '%s\n', opencommand);

    %           loop to write out the node coordinates for each timestep
    %           specified above
                for j = 1:timesteps;

                    node_disp_name = [num2str(i),'-',num2str(j)];
                    command_name = ['output "',sim_bundle_file_path,...
                        'Node_out/',node_disp_name,'" ',num2str(j),...
                        ' 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1.000000'];

                    fprintf(fid, '%s\n', command_name);

                end
                    fprintf(fid, 'exit\n');

                    fclose(fid);
                    command = ['lspp45',' ',commandfile, ' -nographics &'];
                    status = system(command);
            end
    end

    %%  Read node coordinates at each simulation at each timestep

    node_displacement = cell(simulations,timesteps);

    %%  Do loop

    for i = 1:simulations
```

```
78          for j= 1:timesteps
79
80              %%
81
82              timesim = [num2str(i),'_',num2str(j)];
83              keyword_path = [sim_bundle_file_path,'Node_out/',timesim];
84
85
86              meshk = fileread(keyword_path);
87              meshk = textscan(meshk,'%s','Delimiter','\n')';
88              meshk = meshk{1};
89
90
91
92               nodename = strfind(meshk, '*NODE');
93               line  = find(not(cellfun('isempty', nodename)));
94               docend = strfind(meshk, '*END');
95               endline  = find(not(cellfun('isempty', docend)));
96
97              nodecoords = meshk(line+1:endline-1);
98
99              nodecoords = cell2mat(cellfun(@str2num,nodecoords,...
100                 'UniformOutput',false));
101
102             node_displacement{i,j} = nodecoords(nodeids,2:4)';
103
104         end
105     end
106
107     save([sim_bundle_file_path,'node_displacement.mat'],'node_displacement');
108
109 else
110     load([sim_bundle_file_path,'node_displacement.mat']);
111
112     for i = 1:length(node_displacement(:,1))
113
114         for j = 1:length(node_displacement(1,:))
115
116             nodess = node_displacement{i,j};
117             node_displacement{i,j} = nodess(:,nodeids);
118
119         end
120
121     end
122
123 end
124
125
126 end
```

A.5 Spec_coeff.m

```
1  function [spectral_coefficients] = Spec_coeff(node_displacement, Evecs)
```

```
2  %Spec_coeff calculates the spectral coefficients
3  %   INPUT:  node_displacement: cell matrix of size [simulations X
4  %           timesteps] where each cell contains the nodal coordinates of the part
5  %           at that specific time/simulation
6  %           Evecs: the eigenvectors to calculate the spectral coefficients
7  %           for
8
9  %   OUTPUT: spectral_coefficients: cell matrix containing the spectral
10 %           coefficients for each simulation at each timestep. Rows =
11 %           simulation, columns = timestep
12
13 %%  Calculate Spectral Coefficients
14
15 timesteps = length(node_displacement(1,:));
16 simulations = length(node_displacement(:,1));
17
18 %   allocating the space for a cell matrix, the contents of the cells will be redefined in
19 %   the next step
20     spectral_coefficients = node_displacement;
21
22     for i = 1:simulations
23         for j = 1:timesteps
24
25         tempt = node_displacement{i,j};
26
27         spectral_coefficients{i,j} = tempt * Evecs;
28
29         clear tempt
30
31         end
32     end
33
34
35
36
37 end
```

A.6 deformation_modes.m

```
1  function [Deformation_modes, mode_sims] = deformation_modes(type, node_displacement, modenum, mesh,
       Spectral_Coefficients)
2  %   deformation_modes:    defines the deformation modes in the spectral
3  %   domain or geometric domain
4  %
5  %   INPUT:  type: Spectral clustering or geometric clustering, input is 'S'
6  %           or 'G', default is 'G' if something else id defined
7  %
8  %           node_displacement: cell matrix of size [simulations X
9  %           timesteps] where each cell contains the nodal coordinates of
10 %           the part at that specific time/simulation
11 %
12 %           modenum: number of deformation modes you want to cluster your
13 %           data in
```

```
14 %
15 %           mesh: struct of the geometry values containing,
16 %               mesh.geometry.TRIV: Faces
17 %               mesh.geometry.X: X coordinates
18 %               mesh.geometry.Y: Y coordinates
19 %               mesh.geometry.Z: Z coordinates
20 %               mesh.geometry.nodenum: number of nodes
21 %
22 %           Spectral_Coefficients:  cell matrix containing the spectral
23 %           coefficients for each simulation at each timestep. Rows =
24 %           simulation, columns = timestep
25 %
26 %   OUTPUT: Deformation_modes:  struct contining the following variables
27 %               Deformation_modes.X:  node coordinates for the mean of each
28 %               modenum when clustering by spectral coefficients the X
29 %               direction
30 %
31 %               Deformation_modes.Y:  node coordinates for the mean of each
32 %               modenum when clustering by spectral coefficients the Y
33 %               direction
34 %
35 %               Deformation_modes.Z:  node coordinates for the mean of each
36 %               modenum when clustering by spectral coefficients the Z
37 %               direction
38 %
39 %           mode_sims:  struct containing the following information
40 %               mode_sims.X:  simulations belonging to each cluster when
41 %               clusting by the coefficients in the X direction
42 %
43 %               mode_sims.Y:  simulations belonging to each cluster when
44 %               clusting by the coefficients in the Y direction
45 %
46 %               mode_sims.Z:  simulations belonging to each cluster when
47 %               clusting by the coefficients in the Z direction
48 %
49 %
50 %   PLOTS:  Deformation clusters plot:  Shows the representative
51 %           deformation for the clusters at the final timestep. The rows
52 %           are clusters created from data in the x, y and z direction. The
53 %           columns are defined by the number of desired deformation modes
54 %
55
56 %%
57
58 %   Reads input to determine if it should cluster by the spectral
59 %   coefficients or geometric deformation, the default is geometric
60 %   deformation
61
62     if type == 'S'
63
64         parameter = Spectral_Coefficients;
65         ytitle = 'Spectral Coefficients';
66         plottitle = 'Clustering in the Spectral Domain';
67         typetitle = 'Coefficients';
68
69     elseif type == 'G'
```

```
70
71          parameter = node_displacement;
72          ytitle = 'Node Id''s';
73          plottitle = 'Clustering in the Geometric Domain';
74          typetitle = 'Displacement';
75
76      else
77
78          parameter = node_displacement;
79          ytitle = 'Node Id''s';
80          plottitle = 'Clustering in the Geometric Domain';
81          typetitle = 'Displacement';
82
83      end
84
85  %   Setting variables
86      timesteps = length(node_displacement(1,:));
87      simulations = length(node_displacement(:,1));
88      node_displacement_afterstart = cell(simulations,timesteps);
89
90
91  %   Calculates the change in the parameter at each simulation/timestep;
92      for i = 1:simulations
93          for j = 1:timesteps
94             node_displacement_afterstart{i,j} = (parameter{i,j} - parameter{1,1});
95          end
96      end
97
98
99
100 %   Organizes the data into x, y and z
101     nodedisp = cell(simulations,3);
102     for i = 1:simulations;
103
104         current = node_displacement_afterstart{i,timesteps};
105
106         nodedisp{i,1} = current(1,:);
107         nodedisp{i,2} = current(2,:);
108         nodedisp{i,3} = current(3,:);
109
110     end
111
112
113 %%  Take the sum of the values in X, Y and Z
114 %   I do this as a parameter for clustering
115
116
117 %   Sum of the values in the X direction
118     X = nodedisp(:,1);
119     X = cell2mat(X);
120     Xsum = sum(X,2);
121
122 %   Sort the sum values by their magnitude
123     [Xsumsort, Xsortorder] = sort(Xsum);
124
125 %   Sum of the values in the Y direction
```

```
126     Y = nodedisp(:,2);
127     Y = cell2mat(Y);
128     Ysum = sum(Y,2);
129 
130 %   Sort the sum values by their magnitude
131     [Ysumsort, Ysortorder] = sort(Ysum);
132 
133 
134 %   Sum of the values in the Z direction
135     Z = nodedisp(:,3);
136     Z = cell2mat(Z);
137     Zsum = sum(Z,2);
138 
139 %   Sort the sum values by their magnitude
140     [Zsumsort, Zsortorder] = sort(Zsum);
141 
142 
143 %% X cluster
144 
145 %   Clusters x into n clusters, where n = modenum
146     idx = clusterdata(Xsumsort,'Maxclust',modenum);
147 
148 %   pre-allocating space for the node coordinate data
149     Deformation_modes.X = cell(timesteps,modenum);
150 
151 %   pre-allocating space for the list of simulations belonging to each
152 %   cluster
153     mode_sims.X = cell(1,3);
154 
155 %   loop that calculates the 'mean' representation of the geometry in the
156 %   cluster
157     for k = 1:modenum;
158 
159 %       finds all values belonging to the cluster
160         simulationstemp = find(idx == k);
161         temp = Xsortorder(simulationstemp);
162 
163 %       all simulations belonging to the cluster are defined and saved in
164 %       mode_sims
165         mode_sims.X{k} = temp;
166 
167 %       pre-allocating space for the node coordinates for all simulations
168 %       in the cluster
169         nodes = cell(length(temp),3);
170 
171 %       loop for all timesteps
172         for h = 1:timesteps;
173 
174 %           loop for all simulations in the cluster
175             for j = 1:length(temp);
176 
177 %               taking the data from the node_displacement matrix for the
178 %               current simulation at the current timestep
179                 current = node_displacement{temp(j),h};
180 
181 %               Saving the coordinates form x, y and z to the corresponding
```

```
182 %                locations in nodes
183                  nodes{j,1} = current(1,:);
184                  nodes{j,2} = current(2,:);
185                  nodes{j,3} = current(3,:);
186              end
187
188 %            taking average of the data for the current timestep for x, y
189 %            and z of each node
190
191 %            average of x
192              X = nodes(:,1);
193              X = cell2mat(X);
194              Xmean = mean(X,1);
195
196 %            average of y
197              Y = nodes(:,2);
198              Y = cell2mat(Y);
199              Ymean = mean(Y,1);
200
201 %            average of z
202              Z = nodes(:,3);
203              Z = cell2mat(Z);
204              Zmean = mean(Z,1);
205
206 %            saving the new average coordinate data to the current timestep
207 %            in Deformation_modes
208              Deformation_modes.X{h,k} = [Xmean; Ymean; Zmean];
209
210          end
211
212      end
213
214 %    the final result from this section is having a single representative
215 %    'simulation' for each deformation mode at each timestep. this is saved
216 %    in the Deformation_modes variable, the simulations that were averaged
217 %    to create the representative part are saved in mode_sims
218
219
220 %%   Plot the deformation modes for clustering in the x direction
221
222      figure('Name','Deformation Modes, Spectral Coefficients','NumberTitle','off','pos',[100 100 1000 500])
             ;
223
224 %    node coordinates of the undeformed geometry, this is used later to
225 %    calculate the values for the colormap
226      position_start = node_displacement{1,1};
227
228 %    loop that plots the representative geometry for each mode/cluster
229      for i = 1:modenum
230
231 %            calculates the vertices and faces for plotting the mesh
232              vertices = Deformation_modes.X{timesteps,i};
233              vertices = vertices';
234              faces = mesh.geometry.TRIV;
235
236 %            triangulation to order the nodes and faces in the correct order
```

```
237 %           to reconstruct the mesh
238             faces2 = triangulation(faces,vertices(:,1),vertices(:,2),vertices(:,3));
239             faces = faces2.ConnectivityList;
240             vertices = faces2.Points;
241
242 %           saving the data into a struct
243             meshfinal.geometry.TRIV = faces;
244             meshfinal.geometry.X = vertices(:,1);
245             meshfinal.geometry.Y = vertices(:,2);
246             meshfinal.geometry.Z = vertices(:,3);
247             meshfinal.geometry.nodenum = length(vertices);
248
249 %           creates the values allocated to each node for the colormap
250 %           takes the absolute node displacement in x,y and z
251             C = abs(vertices - position_start');
252 %           colormap is based off of displacement in the z direction
253             map = C(:,3);
254
255 %           colormap colors defined as plasma for proper grayscale
256 %           conversion
257             cm = plasma();
258             colormap(cm);
259
260 %           plots the mode the row corresponding to the coordinates used
261 %           for clustering, and column corresponding to the mode number
262             subplot(3,modenum,i);
263
264                 trimesh(meshfinal.geometry.TRIV, meshfinal.geometry.X, meshfinal.geometry.Y, meshfinal.
                        geometry.Z, map, ...
265                     'EdgeColor', 'interp', 'FaceColor', 'interp');
266
267                 view(3);
268                 axis equal;
269                 axis off;
270
271                 lighting gouraud;
272                 shading interp ;
273                 camlight(-37.5,30);
274
275                 title(sprintf('%g, %s', i, 'X'), ...
276                 'interpreter','latex','FontSize',16);
277
278     end
279
280 %   transforms the data before saving
281     Deformation_modes.X = Deformation_modes.X';
282
283
284 %% Y cluster
285
286 %   Clusters y into n clusters, where n = modenum
287     idx = clusterdata(Ysumsort,'Maxclust',modenum);
288
289 %   pre-allocating space for the node coordinate data
290     Deformation_modes.Y = cell(timesteps,modenum);
291
```

```
292 %   pre-allocating space for the list of simulations belonging to each
293 %   cluster
294     mode_sims.Y = cell(1,3);
295
296 %   loop that calculates the 'mean' representation of the geometry in the
297 %   cluster
298     for k = 1:modenum;
299
300 %       finds all values belonging to the cluster
301         simulationstemp = find(idx == k);
302         temp = Ysortorder(simulationstemp);
303
304 %       all simulations belonging to the cluster are defined and saved in
305 %       mode_sims
306         mode_sims.Y{k} = temp;
307
308 %       pre-allocating space for the node coordinates for all simulations
309 %       in the cluster
310         nodes = cell(length(temp),3);
311
312 %       loop for all timesteps
313         for h = 1:timesteps;
314
315 %           loop for all simulations in the cluster
316             for j = 1:length(temp);
317
318 %               taking the data from the node_displacement matrix for the
319 %               current simulation at the current timestep
320                 current = node_displacement{temp(j),h};
321
322 %               Saving the coordinates form x, y and z to the corresponding
323 %               locations in nodes
324                 nodes{j,1} = current(1,:);
325                 nodes{j,2} = current(2,:);
326                 nodes{j,3} = current(3,:);
327             end
328
329 %           taking average of the data for the current timestep for x, y
330 %           and z of each node
331
332 %           average of x
333             X = nodes(:,1);
334             X = cell2mat(X);
335             Xmean = mean(X,1);
336
337 %           average of y
338             Y = nodes(:,2);
339             Y = cell2mat(Y);
340             Ymean = mean(Y,1);
341
342 %           average of z
343             Z = nodes(:,3);
344             Z = cell2mat(Z);
345             Zmean = mean(Z,1);
346
347 %           saving the new average coordinate data to the current timestep
```

```
348 %           in Deformation_modes
349             Deformation_modes.Y{h,k} = [Xmean; Ymean; Zmean];
350
351         end
352
353     end
354
355 %   the final result from this section is having a single representative
356 %   'simulation' for each deformation mode at each timestep. this is saved
357 %   in the Deformation_modes variable, the simulations that were averaged
358 %   to create the representative part are saved in mode_sims
359
360
361 %% Plot the deformation modes for clustering in the y direction
362
363 %   loop that plots the representative geometry for each mode/cluster
364     for i = 1:modenum
365
366 %           calculates the vertices and faces for plotting the mesh
367             vertices = Deformation_modes.Y{timesteps,i};
368             vertices = vertices';
369             faces = mesh.geometry.TRIV;
370
371 %           triangulation to order the nodes and faces in the correct order
372 %           to reconstruct the mesh
373             faces2 = triangulation(faces,vertices(:,1),vertices(:,2),vertices(:,3));
374             faces = faces2.ConnectivityList;
375             vertices = faces2.Points;
376
377 %           saving the data into a struct
378             meshfinal.geometry.TRIV = faces;
379             meshfinal.geometry.X = vertices(:,1);
380             meshfinal.geometry.Y = vertices(:,2);
381             meshfinal.geometry.Z = vertices(:,3);
382             meshfinal.geometry.nodenum = length(vertices);
383
384 %           creates the values allocated to each node for the colormap
385 %           takes the absolute node displacement in x,y and z
386             C = abs(vertices - position_start');
387 %           colormap is based off of displacement in the z direction
388             map = C(:,3);
389
390 %           colormap colors defined as plasma for proper grayscale
391 %           conversion
392             cm = plasma();
393             colormap(cm);
394
395 %           plots the mode the row corresponding to the coordinates used
396 %           for clustering, and column corresponding to the mode number
397             subplot(3,modenum,i+modenum);
398
399                 trimesh(meshfinal.geometry.TRIV, meshfinal.geometry.X, ...
400                     meshfinal.geometry.Y, meshfinal.geometry.Z, map, ...
401                     'EdgeColor', 'interp', 'FaceColor', 'interp');
402
403                 view(3);
```

```
404                 axis equal;
405                 axis off;
406
407                 lighting gouraud;
408                 shading interp ;
409                 camlight(-37.5,30);
410
411                 title(sprintf('%g, %s', i, 'Y'), ...
412                 'interpreter','latex','FontSize',16);
413
414     end
415
416 %   transforms the data before saving
417     Deformation_modes.Y = Deformation_modes.Y';
418
419
420 %% Z cluster
421
422 %   Clusters z into n clusters, where n = modenum
423     idx = clusterdata(Zsumsort,'Maxclust',modenum);
424
425 %   pre-allocating space for the node coordinate data
426     Deformation_modes.Z = cell(timesteps,modenum);
427
428 %   pre-allocating space for the list of simulations belonging to each
429 %   cluster
430     mode_sims.Z = cell(1,3);
431
432 %   loop that calculates the 'mean' representation of the geometry in the
433 %   cluster
434     for k = 1:modenum;
435
436 %       finds all values belonging to the cluster
437         simulationstemp = find(idx == k);
438         temp = Zsortorder(simulationstemp);
439
440 %       all simulations belonging to the cluster are defined and saved in
441 %       mode_sims
442         mode_sims.Z{k} = temp;
443
444 %       pre-allocating space for the node coordinates for all simulations
445 %       in the cluster
446         nodes = cell(length(temp),3);
447
448 %       loop for all timesteps
449         for h = 1:timesteps;
450
451 %           loop for all simulations in the cluster
452             for j = 1:length(temp);
453
454 %               taking the data from the node_displacement matrix for the
455 %               current simulation at the current timestep
456                 current = node_displacement{temp(j),h};
457
458 %               Saving the coordinates form x, y and z to the corresponding
459 %               locations in nodes
```

```
460                     nodes{j,1} = current(1,:);
461                     nodes{j,2} = current(2,:);
462                     nodes{j,3} = current(3,:);
463                 end
464
465 %               taking average of the data for the current timestep for x, y
466 %               and z of each node
467
468 %               average of x
469                 X = nodes(:,1);
470                 X = cell2mat(X);
471                 Xmean = mean(X,1);
472
473 %               average of y
474                 Y = nodes(:,2);
475                 Y = cell2mat(Y);
476                 Ymean = mean(Y,1);
477
478 %               average of z
479                 Z = nodes(:,3);
480                 Z = cell2mat(Z);
481                 Zmean = mean(Z,1);
482
483 %               saving the new average coordinate data to the current timestep
484 %               in Deformation_modes
485                 Deformation_modes.Z{h,k} = [Xmean; Ymean; Zmean];
486
487             end
488
489         end
490
491 %   the final result from this section is having a single representative
492 %   'simulation' for each deformation mode at each timestep. this is saved
493 %   in the Deformation_modes variable, the simulations that were averaged
494 %   to create the representative part are saved in mode_sims
495
496
497 %%  Plot the deformation modes for clustering in the y direction
498
499 %   loop that plots the representative geometry for each mode/cluster
500     for i = 1:modenum
501
502 %               calculates the vertices and faces for plotting the mesh
503                 vertices = Deformation_modes.Z{timesteps,i};
504                 vertices = vertices';
505                 faces = mesh.geometry.TRIV;
506
507 %               triangulation to order the nodes and faces in the correct order
508 %               to reconstruct the mesh
509                 faces2 = triangulation(faces,vertices(:,1),vertices(:,2),...
510                     vertices(:,3));
511                 faces = faces2.ConnectivityList;
512                 vertices = faces2.Points;
513
514 %               saving the data into a struct
515                 meshfinal.geometry.TRIV = faces;
```

```
516             meshfinal.geometry.X = vertices(:,1);
517             meshfinal.geometry.Y = vertices(:,2);
518             meshfinal.geometry.Z = vertices(:,3);
519             meshfinal.geometry.nodenum = length(vertices);
520
521 %           creates the values allocated to each node for the colormap
522 %           takes the absolute node displacement in x,y and z
523             C = abs(vertices - position_start');
524 %           colormap is based off of displacement in the z direction
525             map = C(:,3);
526
527 %           colormap colors defined as plasma for proper grayscale
528 %           conversion
529             cm = plasma();
530             colormap(cm);
531
532 %           plots the mode the row corresponding to the coordinates used
533 %           for clustering, and column corresponding to the mode number
534             subplot(3,modenum,i+2*modenum);
535
536                 trimesh(meshfinal.geometry.TRIV, meshfinal.geometry.X, ...
537                     meshfinal.geometry.Y, meshfinal.geometry.Z, map, ...
538                     'EdgeColor', 'interp', 'FaceColor', 'interp');
539
540                 view(3);
541                 axis equal;
542                 axis off;
543
544                 lighting gouraud;
545                 shading interp ;
546                 camlight(-37.5,30);
547
548                 title(sprintf('%g, %s', i, 'Z'), ...
549                 'interpreter','latex','FontSize',16);
550
551     end
552
553 %   transforms the data before saving
554     Deformation_modes.Z = Deformation_modes.Z';
555
556
557
558     end
```

A.7 time_frequency_cluster.m

```
1  function [coeff] = time_frequency_cluster( Spectral_Coefficients_Calc, Spectral_Coefficients_New,
       desired_deformation_mode, mode_sims, colorscale)
2  %Plots Spectral coefficients from laplace beltrami operator
3  %   INPUT:  Spectral_Coefficients_Calc: cell matrix containing the spectral
4  %                                       coefficients for all the simulations
5  %                                       for each simulation at each timestep.
6  %                                       Rows = simulation, columns = timestep
```

```
%
%           Spectral_Coefficients_New:  cell matrix containing the spectral
%                                       coefficients for the deformation modes representative parts
%                                       for each representative simulation at each timestep.
%                                       Rows = simulation, columns = timestep
%
%           desired_deformation_mode:   integer corresponding to the
%                                       deformation mode being considerend in the analysis
%
%           mode_sims:  struct containing the following information
%                       mode_sims.X:  simulations belonging to each cluster when clusting by
%                       the coefficients in the X direction
%
%                       mode_sims.Y:  simulations belonging to each cluster when clusting by
%                       the coefficients in the Y direction
%
%                       mode_sims.Z:  simulations belonging to each cluster when clusting by
%                       the coefficients in the Z direction
%
%           colorscale: 'B' for binary colorscale, and 'M' for multi-color
%                       colorscale
%
%   OUTPUT: coeff:  struct containing the following information
%
%                   coeff.X:    X coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 1 = x
%                   first row is the coefficients
%
%                   coeff.Y:    Y coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 2 = y
%                   first row is the coefficients
%
%                   coeff.Z:    Z coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 3 = z
%                   first row is the coefficients
%
%   PLOTS:  simulation frequency plot: The y-axis is the spectral
%           coefficients ordered by the magnitude of their corresponding
%           eigenvalue,the x-axis is the simulations belonging to the
%           cluster for the desired_deformation_mode
%
%           time frequency plot: The y-axis is the spectral
%           coefficients ordered by the magnitude of their corresponding
%           eigenvalue,the x-axis is the timesteps for the representitive
%           part for the desired_deformation_mode
%
%
```

```
63
64  %%   Data
65
66      timesteps = length(Spectral_Coefficients_Calc(1,:));
67      simulations = length(Spectral_Coefficients_Calc(:,1));
68
69  %%  Calculating the mean value of the coefficients at the final timestep for all simulations
70  %   Taking the mean over X, Y, and Z as well
71
72  %   Taking the mean over all simulations in bundle
73      meanval_coeffs = Spectral_Coefficients_Calc(:,timesteps);
74
75      meanval = cell2mat(meanval_coeffs);
76
77      m = mean(mean(mean(meanval)));
78
79  %%  Importing the spectral coefficients for the desired deformation mode
80
81  %   coefficients at desired deformation mode final timestep
82      finaltime_coeffs = Spectral_Coefficients_New(desired_deformation_mode,timesteps);
83
84      finaltime = cell2mat(finaltime_coeffs);
85
86  %%  start_coeff
87
88      starttime_coeffs = Spectral_Coefficients_New(desired_deformation_mode,1);
89
90      starttime = cell2mat(starttime_coeffs);
91  %%
92
93  %   Allocating space for the future output(coeff) of this function, values
94  %   will change later according to values specified below
95      coeff.X = finaltime;
96      coeff.Y = finaltime;
97      coeff.Z = finaltime;
98
99  %   Subtracting the mean of the coefficients for all simulations in this
100 %   simulation bundle from the coefficients of the final timestep in the
101 %   desired deformation mode
102     finaltimev = abs(finaltime - m);
103
104     %%
105
106 %   Sorting the values of X, Y, and Z by magnitude to determine which
107 %   coefficients should be used to scale the corresponding eigenvectors,
108 %   this eliminated the X, Y, and Z components and considers just the
109 %   coefficient deviating furthest from the mean.
110     finaltimex = finaltimev(1,:);
111     finaltimey = finaltimev(2,:);
112     finaltimez = finaltimev(3,:);
113
114 %   sorting the values based on the magnitude of the coefficient to
115 %   determine which coefficients have the highest deviation from the mean
116     [magx ordermagx] = sort(finaltimex, 'descend');
117     [magy ordermagy] = sort(finaltimey, 'descend');
118     [magz ordermagz] = sort(finaltimez, 'descend');
```

```
119
120 %   Filling in the values allocated above, first row is the coefficients
121 %   sorted by magnitude, second row is the value corresponding to that
122 %   coefficient and the eigenvector that should be scaled by it, the thid
123 %   row indicates the direction in which the the vector should be scaled,
124 %   x,y or z
125     coeff.X(1,:) = magx;
126     coeff.X(2,:) = ordermagx;
127     coeff.X(3,:) = 1;
128
129     coeff.Y(1,:) = magy;
130     coeff.Y(2,:) = ordermagy;
131     coeff.Y(3,:) = 2;
132
133     coeff.Z(1,:) = magz;
134     coeff.Z(2,:) = ordermagz;
135     coeff.Z(3,:) = 3;
136
137 %% Preparing data for plotting
138
139 %   Coefficients belonging to the the deformation mode representative part at all timesteps
140     finaltime_coeffs2 = Spectral_Coefficients_New(desired_deformation_mode,:);
141
142 %   pre-allocating data for matrices plotted in the time frequency plot
143     NewX = zeros(timesteps,length(coeff.X(1,:)));
144     NewY = zeros(timesteps,length(coeff.X(1,:)));
145     NewZ = zeros(timesteps,length(coeff.X(1,:)));
146
147 %       loop over time steps
148         for i = 1:length(finaltime_coeffs2)
149
150 %           separate the data by the x, y and z directions
151             coeffstemp = finaltime_coeffs2{1,i};
152             NewX(i,:) = coeffstemp(1,:);
153             NewY(i,:) = coeffstemp(2,:);
154             NewZ(i,:) = coeffstemp(3,:);
155
156         end
157
158 %   subtract the mean from each matrix and take the absolute value so
159 %   values will be ranked by their magnitude
160     NewX = abs(NewX - m);
161     NewY = abs(NewY - m);
162     NewZ = abs(NewZ - m);
163
164
165 %% determine the colorscale for the plot
166
167 %   calculate the mean of the coefficients in x, y and z
168     coeffmean.X = mean(coeff.X(1,:));
169     coeffmean.Y = mean(coeff.Y(1,:));
170     coeffmean.Z = mean(coeff.Z(1,:));
171
172 %   pre-allocating the matrices for the plot
173     cx = zeros(size(NewX'));
174     cy = zeros(size(NewX'));
```

```
175     cz = zeros(size(NewX'));
176
177 %   if the colorscale is binary make everything above the mean = 1 and
178 %   everything else =0
179     if colorscale == 'B';
180
181 %       x plot
182 %       find all values in the matrix that are greater than the mean of the
183 %       data
184         import.X = find(NewX' >= (coeffmean.X));
185
186 %       make all values above the mean equal to 1
187         cx(import.X) = 1;
188
189 %       y plot
190 %       find all values in the matrix that are greater than the mean of the
191 %       data
192         import.Y = find(NewY' >= (coeffmean.Y));
193
194 %       make all values above the mean equal to 1
195         cy(import.Y) = 1;
196
197 %       z plot
198 %       find all values in the matrix that are greater than the mean of the
199 %       data
200         import.Z = find(NewZ' >= (coeffmean.Z));
201
202 %       make all values above the mean equal to 1
203         cz(import.Z) = 1;
204
205 %   if colorscale is not binary then set all values above the mean to 0.5
206 %   and all values one standard deviation above the mean equal to 0.75 and
207 %   all values 2 standard deviations above the mean equal to 1
208     else
209
210 %       calculate the standard deviations for x, y and z
211         stdeviation.X = std(coeff.X(1,:));
212         stdeviation.Y = std(coeff.Y(1,:));
213         stdeviation.Z = std(coeff.Z(1,:));
214
215 %       x plot
216 %       find all values in the matrix that are greater than the mean of the
217 %       data
218         import.X = find(NewX' >= (coeffmean.X));
219         cx(import.X) = 0.5;
220
221 %       all values greater than one standard deviation above the mean
222         import2.X = find(NewX' >= (coeffmean.X + stdeviation.X));
223         cx(import2.X) = 0.75;
224
225 %       all values greater than two standard deviations above the mean
226         import3.X = find(NewX' >= (coeffmean.X + 2*stdeviation.X));
227         cx(import3.X) = 1;
228
229 %       y plot
230 %       find all values in the matrix that are greater than the mean of the
```

```
231 %       data
232         import.Y = find(NewY' >= (coeffmean.Y));
233         cy(import.Y) = 0.5;
234
235 %       all values greater than one standard deviation above the mean
236         import2.Y = find(NewY' >= (coeffmean.Y + stdeviation.Y));
237         cy(import2.Y) = 0.75;
238
239 %       all values greater than two standard deviations above the mean
240         import3.Y = find(NewY' >= (coeffmean.Y + 2*stdeviation.Y));
241         cy(import3.Y) = 1;
242
243 %       z plot
244 %       find all values in the matrix that are greater than the mean of the
245 %       data
246         import.Z = find(NewZ' >= (coeffmean.Z));
247         cz(import.Z) = 0.5;
248
249 %       all values greater than one standard deviation above the mean
250         import2.Z = find(NewZ' >= (coeffmean.Z + stdeviation.Z));
251         cz(import2.Z) = 0.75;
252
253 %       all values greater than two standard deviations above the mean
254         import3.Z = find(NewZ' >= (coeffmean.Z + 2*stdeviation.Z));
255         cz(import3.Z) = 1;
256
257     end
258
259
260 %% plot the time-frequency plot
261
262     TF_Plot_Difference = figure('Name','Time Frequency','NumberTitle','off','pos',[100 100 1500 400]);
263
264 %   colormap for proper grayscale conversion
265     cm = plasma();
266     colormap(cm);
267
268 %   subplot for x
269     px = subplot(1,3,1);
270     imagesc(cx)
271     axis xy
272     title('X Coefficients','interpreter','latex','FontSize',16);
273     ylabel('Spectral Coefficients','interpreter','latex','FontSize',16);
274
275 %   subplot for y
276     py = subplot(1,3,2);
277     imagesc(cy)
278     axis xy
279     ax = gca;
280     ax.YAxis.TickValues = [];
281     title('Y Coefficients','interpreter','latex','FontSize',16);
282     xlabel('Time','interpreter','latex','FontSize',16);
283
284 %   subplot for z
285     pz = subplot(1,3,3);
286     imagesc(cz)
```

```
287      axis xy
288      ax = gca;
289      ax.YAxis.TickValues = [];
290      title('Z Coefficients','interpreter','latex','FontSize',16);
291
292  %%  Preparing data for plotting
293
294  %   Coefficients belonging to the the deformation mode representative part at all timesteps
295      finaltime_coeffs2 = Spectral_Coefficients_Calc(mode_sims{desired_deformation_mode},timesteps)';
296
297  %   pre-allocating data for matrices plotted in the time frequency plot
298      NewX = zeros(length(mode_sims{desired_deformation_mode}),length(coeff.X(1,:)));
299      NewY = zeros(length(mode_sims{desired_deformation_mode}),length(coeff.X(1,:)));
300      NewZ = zeros(length(mode_sims{desired_deformation_mode}),length(coeff.X(1,:)));
301
302  %       loop over time steps
303          for i = 1:length(finaltime_coeffs2)
304
305  %           separate the data by the x, y and z directions
306              coeffstemp = finaltime_coeffs2{1,i};
307              NewX(i,:) = coeffstemp(1,:);
308              NewY(i,:) = coeffstemp(2,:);
309              NewZ(i,:) = coeffstemp(3,:);
310
311          end
312
313  %   subtract the mean from each matrix and take the absolute value so
314  %   values will be ranked by their magnitude
315      NewX = abs(NewX - m);
316      NewY = abs(NewY - m);
317      NewZ = abs(NewZ - m);
318
319
320  %%  determine the colorscale for the plot
321
322  %   calculate the mean of the coefficients in x, y and z
323      coeffmean.X = mean(coeff.X(1,:));
324      coeffmean.Y = mean(coeff.Y(1,:));
325      coeffmean.Z = mean(coeff.Z(1,:));
326
327  %   pre-allocating the matrices for the plot
328      cx = zeros(size(NewX'));
329      cy = zeros(size(NewX'));
330      cz = zeros(size(NewX'));
331
332  %   if the colorscale is binary make everything above the mean = 1 and
333  %   everything else =0
334      if colorscale == 'B';
335
336  %       x plot
337  %       find all values in the matrix that are greater than the mean of the
338  %       data
339          import.X = find(NewX' >= (coeffmean.X));
340
341  %       make all values above the mean equal to 1
342          cx(import.X) = 1;
```

```
343
344 %       y plot
345 %       find all values in the matrix that are greater than the mean of the
346 %       data
347         import.Y = find(NewY' >= (coeffmean.Y));
348
349 %       make all values above the mean equal to 1
350         cy(import.Y) = 1;
351
352 %       z plot
353 %       find all values in the matrix that are greater than the mean of the
354 %       data
355         import.Z = find(NewZ' >= (coeffmean.Z));
356
357 %       make all values above the mean equal to 1
358         cz(import.Z) = 1;
359
360 %   if colorscale is not binary then set all values above the mean to 0.5
361 %   and all values one standard deviation above the mean equal to 0.75 and
362 %   all values 2 standard deviations above the mean equal to 1
363     else
364
365 %       calculate the standard deviations for x, y and z
366         stdeviation.X = std(coeff.X(1,:));
367         stdeviation.Y = std(coeff.Y(1,:));
368         stdeviation.Z = std(coeff.Z(1,:));
369
370 %       x plot
371 %       find all values in the matrix that are greater than the mean of the
372 %       data
373         import.X = find(NewX' >= (coeffmean.X));
374         cx(import.X) = 0.5;
375
376 %       all values greater than one standard deviation above the mean
377         import2.X = find(NewX' >= (coeffmean.X + stdeviation.X));
378         cx(import2.X) = 0.75;
379
380 %       all values greater than two standard deviations above the mean
381         import3.X = find(NewX' >= (coeffmean.X + 2*stdeviation.X));
382         cx(import3.X) = 1;
383
384 %       y plot
385 %       find all values in the matrix that are greater than the mean of the
386 %       data
387         import.Y = find(NewY' >= (coeffmean.Y));
388         cy(import.Y) = 0.5;
389
390 %       all values greater than one standard deviation above the mean
391         import2.Y = find(NewY' >= (coeffmean.Y + stdeviation.Y));
392         cy(import2.Y) = 0.75;
393
394 %       all values greater than two standard deviations above the mean
395         import3.Y = find(NewY' >= (coeffmean.Y + 2*stdeviation.Y));
396         cy(import3.Y) = 1;
397
398 %       z plot
```

```
399 %       find all values in the matrix that are greater than the mean of the
400 %       data
401         import.Z = find(NewZ' >= (coeffmean.Z));
402         cz(import.Z) = 0.5;
403
404 %       all values greater than one standard deviation above the mean
405         import2.Z = find(NewZ' >= (coeffmean.Z + stdeviation.Z));
406         cz(import2.Z) = 0.75;
407
408 %       all values greater than two standard deviations above the mean
409         import3.Z = find(NewZ' >= (coeffmean.Z + 2*stdeviation.Z));
410         cz(import3.Z) = 1;
411
412     end
413
414 %% Plotting the simulation frequency plot
415
416     SF_Plot_Difference = figure('Name','Time Frequency','NumberTitle','off','pos',[50 50 1500 400]);
417
418 %   colormap for proper grayscale conversion
419     cm = plasma();
420     colormap(cm);
421
422 %   subplot for x
423     px = subplot(1,3,1);
424     imagesc(cx)
425     axis xy
426     title('X Coefficients','interpreter','latex','FontSize',16);
427     ylabel('Spectral Coefficients','interpreter','latex','FontSize',16);
428
429 %   subplot for y
430     py = subplot(1,3,2);
431     imagesc(cy)
432     axis xy
433     ax = gca;
434     ax.YAxis.TickValues = [];
435     title('Y Coefficients','interpreter','latex','FontSize',16);
436     xlabel('Time','interpreter','latex','FontSize',16);
437
438 %   subplot for z
439     pz = subplot(1,3,3);
440     imagesc(cz)
441     axis xy
442     ax = gca;
443     ax.YAxis.TickValues = [];
444     title('Z Coefficients','interpreter','latex','FontSize',16);
445
446
447 end
```

A.8 important_eigenvectors.m

```
1 function [Evecmode, Evecs_scaled, Eval_important] = important_eigenvectors(coeff, Evecs,
```

```
    Spectral_Coefficients_New, desired_deformation_mode, limit)
%Calculates the important eigenvectors
%
%   INPUT:  coeff:  struct containing the following information
%
%                   coeff.X:    X coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 1 = x
%
%                   coeff.Y:    Y coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 2 = y
%
%                   coeff.Z:    Z coefficients, the first row is the
%                   coefficients sorted by magnitude, the second row is the
%                   value corresponding the the above coefficient and the
%                   third row indicated the direction in which the
%                   eigenvector should be scaled, 3 = z
%
%           Evecs:  Eigenvectors
%
%           Spectral_Coefficients_New:  cell matrix containing the spectral
%                                       coefficients for the deformation modes representative parts
%                                       for each representative simulation at each timestep.
%                                       Rows = simulation, columns = timestep
%
%           desired_deformation_mode:   integer corresponding to the
%                                       deformation mode being considerend in the analysis
%
%           limit:  This determines what values are deemend important, the
%                   value corresponds to the number of standard deviations above the
%                   mean the important eigenvectors are considered at
%
%   OUTPUT: Evecmode:   struct containing the following information
%
%                       Evecmode.X: The important eigenvectors scaled by
%                                   their coefficients and then added together to
%                                   create 1 eigenvector for the x direction
%
%                       Evecmode.Y: The important eigenvectors scaled by
%                                   their coefficients and then added together to
%                                   create 1 eigenvector for the y direction
%
%                       Evecmode.Z: The important eigenvectors scaled by
%                                   their coefficients and then added together to
%                                   create 1 eigenvector for the z direction
%
%           Evecs_scaled:   struct containing the following information
%
%                           Evecs_scaled.X: Scaled eigenvectors by
%                                           coefficients in the x direction
%                           Evecs_scaled.Y: Scaled eigenvectors by
```

```
57 %                                        coefficients in the y direction
58 %                           Evecs_scaled.Z: Scaled eigenvectors by
59 %                                        coefficients in the z direction
60 %
61 %           Eval_important: struct containing the following information
62 %
63 %                           Eval_important.X:   All coefficients that are greater than
64 %                                               mean + limit*standard deviation in the x direction
65 %                                               are saved as the 'important eigenvectors'
66 %
67 %                           Eval_important.Y:   All coefficients that are greater than
68 %                                               mean + limit*standard deviation in the y direction
69 %                                               are saved as the 'important eigenvectors'
70 %
71 %                           Eval_important.Z:   All coefficients that are greater than
72 %                                               mean + limit*standard deviation in the z direction
73 %                                               are saved as the 'important eigenvectors'
74
75 %% Code
76
77 %  Scales the eigenvectors by the x, y and z coefficients
78     [Evecs_scaled.X] = scale_evecs(coeff.X, Evecs, Spectral_Coefficients_New, desired_deformation_mode);
79     [Evecs_scaled.Y] = scale_evecs(coeff.Y, Evecs, Spectral_Coefficients_New, desired_deformation_mode);
80     [Evecs_scaled.Z] = scale_evecs(coeff.Z, Evecs, Spectral_Coefficients_New, desired_deformation_mode);
81
82 %  X values
83
84 %       calculate the standard deviation and the mean
85         stdabovemean.X = std(coeff.X(1,:));
86         coeffmean.X = mean(coeff.X(1,:));
87
88 %       find all coefficients that are greater than the mean + limit*standard deviation
89         import.X = find(coeff.X(1,:) >= (coeffmean.X + (limit * stdabovemean.X)));
90
91 %       All coefficients that are greater than mean + limit*standard
92 %       deviation are saved as the 'important eigenvectors'
93         Eval_important.X = coeff.X(2,import.X);
94
95 %       filters out only the important scaled eigenvectors and then adds
96 %       them together making a reconstrction from the important
97 %       eigenvectors
98         num.X = Evecs_scaled.X(:,Eval_important.X);
99         Evecmode.X = sum(num.X,2);
100
101
102 %  Y values
103
104 %       calculate the standard deviation and the mean
105         stdabovemean.Y = std(coeff.Y(1,:));
106         coeffmean.Y = mean(coeff.Y(1,:));
107
108 %       find all coefficients that are greater than the mean + limit*standard deviation
109         import.Y = find(coeff.Y(1,:) >= (coeffmean.Y + (limit * stdabovemean.Y)));
110
111 %       All coefficients that are greater than mean + limit*standard
112 %       deviation are saved as the 'important eigenvectors'
```

```
113         Eval_important.Y = coeff.Y(2,import.Y);
114
115 %       filters out only the important scaled eigenvectors and then adds
116 %       them together making a reconstrction from the important
117 %       eigenvectors
118         num.Y = Evecs_scaled.Y(:,Eval_important.Y);
119         Evecmode.Y = sum(num.Y,2);
120
121
122 %   Z values
123
124 %       calculate the standard deviation and the mean
125         stdabovemean.Z = std(coeff.Z(1,:));
126         coeffmean.Z = mean(coeff.Z(1,:));
127
128 %       find all coefficients that are greater than the mean + limit*standard deviation
129         import.Z = find(coeff.Z(1,:) >= (coeffmean.Z + (limit * stdabovemean.Z)));
130
131 %       All coefficients that are greater than mean + limit*standard
132 %       deviation are saved as the 'important eigenvectors'
133         Eval_important.Z = coeff.Z(2,import.Z);
134
135 %       filters out only the important scaled eigenvectors and then adds
136 %       them together making a reconstrction from the important
137 %       eigenvectors
138         num.Z = Evecs_scaled.Z(:,Eval_important.Z);
139         Evecmode.Z = sum(num.Z,2);
140
141
142 end
```

A.8.1 scale_evecs.m

```
1  function [Evecs_scaled] = scale_evecs(coeff, Evecs, Spectral_Coefficients_New, desired_deformation_mode)
2  %%scale_evecs
3  %
4  %   INPUT:  coeff:  coefficients, the first row is the
5  %                   coefficients sorted by magnitude, the second row is the
6  %                   value corresponding the the above coefficient and the
7  %                   third row indicated the direction in which the
8  %                   eigenvector should be scaled, for example 2 = y
9  %
10 %           Evecs:  Eigenvectors
11 %
12 %           Spectral_Coefficients_New:  cell matrix containing the spectral
13 %                                       coefficients for the deformation
14 %                                       modes representative parts for each
15 %                                       representative simulation at each
16 %                                       timestep. Rows = simulation,
17 %                                       columns = timestep
18 %
19 %           desired_deformation_mode:   integer corresponding to the
20 %                                       deformation mode being considerend
21 %                                       in the analysis
```

```
22 %
23 %   OUTPUT: Evecs_scaled:   The eigenvectors scaled by their corresponding
24 %                           coefficients
25
26 %%  Function Start
27
28 %   define number of timesteps
29     timesteps = length(Spectral_Coefficients_New(1,:));
30
31 %   coefficients used for scaling
32     coefficients = Spectral_Coefficients_New{desired_deformation_mode,...
33         timesteps};
34
35 %   reads the first value in the 3rd row to determine which direction to
36 %   scale by, x, y or z.
37     direction = coeff(3,1);
38
39 %   sets the coefficients equal to the coefficients in the correct
40 %   direction
41     coefficients = coefficients(direction,:);
42
43 %   scales each value in the eigenvector by the coefficient, this is done
44 %   for each eigenvector-coefficient pair
45     Evecs_scaled = bsxfun(@times,Evecs,coefficients);
46
47
48
49 end
```

A.9 similarity.m

```
1  function [ similar ] = similarity( node_displacement, A, Eval_important, Evecs )
2  %similarity
3  %
4  %   INPUT:  node_displacement:  cell matrix of size [simulations X
5  %                               timesteps] where each cell contains the nodal coordinates of
6  %                               the part at that specific time/simulation
7  %
8  %           A:  Spectral coefficients for the desired deformation mode at
9  %               the final timestep.
10 %
11 %           Eval_important: struct containing the following information
12 %
13 %                           Eval_important.X:   All coefficients that are greater than
14 %                                               mean + limit*standard deviation in the x direction
15 %                                               are saved as the 'important eigenvectors'
16 %
17 %                           Eval_important.Y:   All coefficients that are greater than
18 %                                               mean + limit*standard deviation in the y direction
19 %                                               are saved as the 'important eigenvectors'
20 %
21 %                           Eval_important.Z:   All coefficients that are greater than
22 %                                               mean + limit*standard deviation in the z direction
```

```
23 %                                          are saved as the 'important eigenvectors'
24 %
25 %           Evecs: Eigenvectors
26 %
27 %   OUTPUT: similar: list of the simulations in the bundle ranked from most
28 %                    similar to least similar.
29
30 %% Function Start
31
32 %   Makes a vector of the values of the 'important' coefficients for x, y
33 %   and z. The 'important' coefficients are the coefficients corresponding
34 %   to the important eigenvectors
35
36     A = [A(1,Eval_important.X) A(2,Eval_important.Y) A(3,Eval_important.Z)];
37
38 %   calculates the spectral coefficients for the simulations in the bundle
39 %   being checked for similarity. The node coordinates fot the simulation
40 %   bundle are stored in node_displacement
41     Spectral_Coefficients = Spec_coeff(node_displacement, Evecs);
42
43 %   set variables to timesteps and simulations.
44     timesteps = length(Spectral_Coefficients(1,:));
45     simulations = length(Spectral_Coefficients(:,1));
46
47 %   preallocated space for the cosine similarity solutions
48     cosine_similarity = zeros(1,simulations);
49
50 %       loop to calculate the cosine similarity for each simulation
51         for i = 1:simulations;
52
53 %           define the vector of 'important' coefficients for the
54 %           simulation being compared to the desired solution
55             B = Spectral_Coefficients{i, timesteps};
56             B =[B(1,Eval_important.X) B(2,Eval_important.Y) B(3,Eval_important.Z)];
57
58 %           preform the cosine similarity between the desired solution and
59 %           the current simulation
60             Adot = A*A';
61             Bdot = B*B';
62             ABdot = A*B';
63             cossim = (ABdot)/(sqrt(Adot)* sqrt(Bdot));
64
65 %           save the solution to the corresponding place in the cosine_similarity vector
66             cosine_similarity(1,i) = cossim;
67
68         end
69
70 %   sort the solutions by most similar to least similar
71     [sorted, similar] = sort(cosine_similarity, 'descend');
72
73 end
```

A.10 reconstruction_comparison_all.m

```
function reconstruction_comparison_all(Spectral_Coefficients_All, deformation_modes, IP, mesh, Evecsall, Eval_importantm)
%%  reconstruction_comparison_all
%
%
%   INPUTS: Spectral_Coefficients_All:
%           deformation_modes:
%           IP:
%           mesh:
%           Evecsall:
%           Eval_importantm:
%
%   PLOTS:    Reconstruction of the part is executed using the coefficients above the mean for each deformation mode.
%             Each row shows the reconstruction of a specific deformation mode.
%             The first column shows the reconstruction when excluding the first IP eigenvectors.
%             The second column shows the reconstruction of the same set of eigenvectors, including the first IP eigenvectors.
%             The eigenvectors above the mean value for each X, Y and Z axes were considered.
%             Some of these eigenvectors were above the mean in multiple axes,
%             however there are n unique eigenvectors included in the displayed reconstructions.
%             The third column shows the reconstruction of all eigenvectors.
%             The fourth column shows the reconstruction if one were to reconstruct using n number of eigenvectors,
%             but instead using the first n eigenvectors ranked by the magnitude of their eigenvalue
%             rather than the magnitude of their coefficients.


%%

plot = figure('Name','Geom Reconstruct','NumberTitle','off','pos',[200 200 2000 1000]);

l = 1;

for m = 1:length(deformation_modes);

    Eval_important = Eval_importantm(m);

    desired_deformation_mode = deformation_modes(m);

    reconstruct_num = [Eval_important.X Eval_important.Y Eval_important.Z];
    reconstruct = unique(reconstruct_num);
    u = length(reconstruct)+IP;

%%

    coefficients = Spectral_Coefficients_All{desired_deformation_mode,20};

    coefficientsX = coefficients(1,:);
    coefficientsY = coefficients(2,:);
    coefficientsZ = coefficients(3,:);

     Evecs_scaledX14 = bsxfun(@times,Evecsall(:,:),coefficientsX);
     Evecs_scaledY14 = bsxfun(@times,Evecsall(:,:),coefficientsY);
```

```
51        Evecs_scaledZ14 = bsxfun(@times,Evecsall(:,:),coefficientsZ);
52
53       NewX14 = sum(Evecs_scaledX14(:,1:IP),2);
54       NewY14 = sum(Evecs_scaledY14(:,1:IP),2);
55       NewZ14 = sum(Evecs_scaledZ14(:,1:IP),2);
56
57
58
59       Evecs_scaledX = Evecs_scaledX14(:,(IP + 1):end);
60       Evecs_scaledY = Evecs_scaledY14(:,(IP + 1):end);
61       Evecs_scaledZ = Evecs_scaledZ14(:,(IP + 1):end);
62
63  %new plots
64
65  meshUpdate1 = mesh;
66  meshUpdate2 = mesh;
67  meshUpdate3 = mesh;
68  meshUpdate4 = mesh;
69
70  NewX = sum(Evecs_scaledX(:,Eval_important.X),2);
71  NewY = sum(Evecs_scaledY(:,Eval_important.Y),2);
72  NewZ = sum(Evecs_scaledZ(:,Eval_important.Z),2);
73
74
75  %plot = figure('Name','Geom Reconstruct','NumberTitle','off','pos',[200 200 2000 1000]);
76
77  subplot(length(deformation_modes),4,(1))
78
79
80  meshUpdate1.geometry.X = NewX;
81  meshUpdate1.geometry.Y = NewY;
82  meshUpdate1.geometry.Z = NewZ;
83
84  trimesh(meshUpdate1.geometry.TRIV, meshUpdate1.geometry.X,meshUpdate1.geometry.Y,meshUpdate1.geometry.Z,
        ...
85                      'EdgeColor', 'interp', 'FaceColor', 'interp');
86
87              view(3);
88              axis equal;
89              axis off;
90
91              lighting gouraud;
92              shading interp ;
93              camlight(-37.5,30);
94
95              title({'Important coefficients',['(excluding 1-',num2str(IP),')']},...
96                  'interpreter','latex','HorizontalAlignment', 'center','FontSize',12);
97
98
99  subplot(length(deformation_modes),4,(1 + 1))
100
101 meshUpdate2.geometry.X = NewX + NewX14;
102 meshUpdate2.geometry.Y = NewY + NewY14;
103 meshUpdate2.geometry.Z = NewZ + NewZ14;
104
105 trimesh(meshUpdate2.geometry.TRIV, meshUpdate2.geometry.X, meshUpdate2.geometry.Y,meshUpdate2.geometry.Z,
```

```
    ...
106                     'EdgeColor', 'interp', 'FaceColor', 'interp');
107
108             view(3);
109             axis equal;
110             axis off;
111
112
113             lighting gouraud;
114             shading interp ;
115             camlight(-37.5,30);
116
117             title({['Important coefficients including 1-',num2str(IP),', '],[num2str(u),' unique
                    eigenvectors']},...
118                 'interpreter','latex','HorizontalAlignment', 'center','FontSize',12);
119
120
121
122 subplot(length(deformation_modes),4,(1 + 2))
123
124 NewXall = sum(Evecs_scaledX14,2);
125 NewYall = sum(Evecs_scaledY14,2);
126 NewZall = sum(Evecs_scaledZ14,2);
127
128 meshUpdate3.geometry.X = NewXall;
129 meshUpdate3.geometry.Y = NewYall;
130 meshUpdate3.geometry.Z = NewZall;
131
132
133 trimesh(meshUpdate3.geometry.TRIV, meshUpdate3.geometry.X, meshUpdate3.geometry.Y, meshUpdate3.geometry.Z,
        ...
134                     'EdgeColor', 'interp', 'FaceColor', 'interp');
135
136             view(3);
137             axis equal;
138             axis off;
139
140
141             lighting gouraud;
142             shading interp ;
143             camlight(-37.5,30);
144
145             title(['First ',num2str(length(Evecsall(1,:))),' coefficients'],...
146                 'interpreter','latex','HorizontalAlignment', 'center','FontSize',12);
147
148
149
150 subplot(length(deformation_modes),4,(1 + 3))
151
152 NewXall = sum(Evecs_scaledX14(:,1:u),2);
153 NewYall = sum(Evecs_scaledY14(:,1:u),2);
154 NewZall = sum(Evecs_scaledZ14(:,1:u),2);
155
156 meshUpdate4.geometry.X = NewXall;
157 meshUpdate4.geometry.Y = NewYall;
158 meshUpdate4.geometry.Z = NewZall;
```

```
159
160
161 trimesh(meshUpdate4.geometry.TRIV, meshUpdate4.geometry.X, meshUpdate4.geometry.Y, meshUpdate4.geometry.Z,
        ...
162                     'EdgeColor', 'interp', 'FaceColor', 'interp');
163
164             view(3);
165             axis equal;
166             axis off;
167
168
169             lighting gouraud;
170             shading interp ;
171             camlight(-37.5,30);
172
173
174             title(['First ',num2str(u),' coefficients'],...
175                 'interpreter','latex','HorizontalAlignment', 'center','FontSize',12);
176
177  l = l+4;
178 end
179
180  cm = plasma();
181  colormap(cm);
182
183 %%  Similarity
184
185
186 Ax = meshUpdate3.geometry.X;
187 Ay = meshUpdate3.geometry.Y;
188 Az = meshUpdate3.geometry.Z;
189
190 Bx = meshUpdate2.geometry.X;
191 By = meshUpdate2.geometry.Y;
192 Bz = meshUpdate2.geometry.Z;
193
194 Cx = meshUpdate4.geometry.X;
195 Cy = meshUpdate4.geometry.Y;
196 Cz = meshUpdate4.geometry.Z;
197
198
199 AB.x = cosineSimilarity(Ax',Bx');
200 AB.y = cosineSimilarity(Ay',By');
201 AB.z = cosineSimilarity(Az',Bz');
202
203 AC.x = cosineSimilarity(Ax',Cx');
204 AC.y = cosineSimilarity(Ay',Cy');
205 AC.z = cosineSimilarity(Az',Cz');
206
207 end
```

A.11 plot_final_timestep.m

```
function [plot] = plot_final_timestep(Desired_Simulations, mesh, node_displacement)


timesteps = length(node_displacement(1,:));
position_start = node_displacement{1,1};
simulations = length(Desired_Simulations);
%%

    plot = figure('Name','Deformation Modes','NumberTitle','off','pos',[100 100 1000 500]);


    for i = 1:simulations

            sub = simulations/5;
            sub = ceil(sub);


            j = Desired_Simulations(i);

            vertices = node_displacement{j,timesteps};
            vertices = vertices';

            faces = mesh.geometry.TRIV;

            faces2 = triangulation(faces,vertices(:,1),vertices(:,2),vertices(:,3));

            faces = faces2.ConnectivityList;
            vertices = faces2.Points;

            meshfinal.geometry.TRIV = faces;
            meshfinal.geometry.X = vertices(:,1);
            meshfinal.geometry.Y = vertices(:,2);
            meshfinal.geometry.Z = vertices(:,3);
            meshfinal.geometry.nodenum = length(vertices);

            C = abs(vertices - position_start');
            normC = C - min(C(:));
            normC = normC ./ max(normC(:));

            map = normC(:,3);



            subplot(sub,5,i);

                trimesh(meshfinal.geometry.TRIV, meshfinal.geometry.X,meshfinal.geometry.Y,meshfinal.
                    geometry.Z, map, ...
                    'EdgeColor', 'interp', 'FaceColor', 'interp');

                view(3);
                axis equal;
                axis off;

                lighting gouraud;
                shading interp ;
                camlight(-37.5,30);
```

```
56
57                  title(sprintf('%g', j), ...
58                  'interpreter','latex','FontSize',16);
59
60                  cm = plasma();
61                  colormap(cm);
62
63
64      end
```